Desarrollo de una plataforma web para el análisis estadístico y el apoyo a la toma de decisiones empresariales en el sector retail colombiano

Daniel Alejandro León Ortiz, Jhon Danilo Rincón Maldonado y Yefferson Olivos Rodríguez

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Director

Urbano Eliécer Gómez Prada

Doctor en Tecnología Educativa

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Ingeniería de Sistemas

Bucaramanga

2025

Agradecimientos

Al concluir este proyecto, deseamos manifestar nuestro más profundo agradecimiento a todas las personas que contribuyeron a que hoy alcance esta meta.

En primer lugar, agradecemos sinceramente al profesor Urbano Eliécer Gómez Prada, quien, como director de este trabajo, nos ofreció su orientación, acompañamiento y valiosas sugerencias, mostrando un compromiso constante en cada fase del proceso. Su guía fue esencial para el desarrollo académico y técnico de este proyecto.

También expresamos nuestro agradecimiento al profesor Luis Carlos Gómez Flórez, por ser un ejemplo de excelencia, por su enseñanza constante y su respaldo a lo largo de nuestra formación y en este proyecto. Su influencia ha marcado significativamente nuestro crecimiento tanto profesional como personal.

De misma forma agradecemos a Grupo APL Ingeniería LTDA, por darnos la oportunidad de realizar nuestra práctica empresarial, permitiéndonos poner en práctica lo aprendido y enfrentar desafíos reales que enriquecieron nuestro proceso formativo.

Asimismo, agradecemos a todos nuestros familiares, docentes, personal administrativo y compañeros que, de distintas maneras, contribuyeron en nuestra trayectoria, ya fuera con su enseñanza, su apoyo o una palabra oportuna en los momentos necesarios. Este logro también les pertenece.

Tabla de Contenido

Introducción 17

1 Planteamiento y Justificación del Problema 19

2 Objetivos 21

2.1 Objetivo General 21

2.2 Objetivos Específicos 21

3 Marco de referencia 22

3.1 Marco contextual 22

3.1.1 Indicadores Clave de Desempeño (KPIs) 22

3.1.2 Modelos Predictivos 22

3.1.2.1 ARIMA 23

3.1.2.2 Sarima 24

3.1.2.3 Prophet 25

3.1.3 Arquitectura multicliente 25

3.1.4 Arquitectura limpia 26

3.1.5 Proceso ETL (Extracción, Transformación y Carga) 28

3.2 Marco tecnológico 29

3.2.1 Ecosistema tecnológico del frontend 29

3.2.1.1 Frameworks y bibliotecas de interfaz de usuario 29

3.2.1.2 Lenguajes y tipado estático 30

3.2.1.3 Construcción y gestión de dependencias 31

3.2.1.4 Comunicación con APIs y servicios externos 31

3.2.1.5 Calidad y estilo del código 32

3.2.1.6 Pruebas y aseguramiento de calidad 32

3.2.2 Ecosistema tecnológico del backend..... 33

3.2.2.1 Lenguaje y plataforma de desarrollo..... 33

3.2.2.2 Gestión y acceso a datos 34

3.2.2.3 Seguridad y calidad del software 35

3.2.3 Tecnologías para análisis de datos y modelos estadísticos 35

3.2.4 Herramientas Complementarias de Desarrollo 37

3.2.4.1 Cursor……..... 37

3.2.4.2 Azure devops……..... 37

3.2.4.3 Visual Studio……..... 38

3.2.4.4 SQL Server Management Studio 20 38

3.2.4.5 Postman……..... 38

3.3 Antecedentes del tema 39

3.3.1 Google Analytics……..... 39

3.3.2 Power BI…….. 39

3.3.3 Tableau……..... 40

4 Metodología 41

4.1 Scrumban…. 41

4.1.1 Conceptos de Scrum…… 41

4.1.1.1 Product Owner…….. 42

4.1.1.2 Sprints…….. 42

4.1.1.3 Sprint Planning……..... 42

4.1.1.4 Sprint Review……..... 43

4.1.1.5 Sprint Retrospective……. 43

4.1.1.6 Product Backlog……. 43

4.1.1.7 Sprint Backlog……. 44

4.1.2 Conceptos de Kanban……. 44

4.1.2.1 Tablero Kanban……. 44

4.1.2.2 Tickets Kanban…… 45

4.1.2.3 Work in progress……. 45

4.2 Extreme Programming (XP) 45

4.2.1 Refactorización…… 46

4.2.2 Estándares de código……. 46

4.2.3 Diseño simple……. 46

4.2.4 Metáfora del sistema……. 47

5 Desarrollo del Proyecto 48

5.1 Análisis y Requerimientos 48

5.1.1 Requerimientos Funcionales 48

5.1.2 Requerimientos no funcionales 55

5.2 Diagramas UML 58

5.2.1 Diagrama de Casos de Uso 58

5.2.2 Diagramas de Actividades 60

5.3 Arquitectura del Software 80

5.3.1 Arquitectura del Frontend 81

5.3.2 Arquitectura del Backend de Gestión y control 83

5.3.3 Arquitectura del Backend para Cálculo de KPIs 84

5.3.4 Multicliente……. 85

5.3.5 Proceso ETL……..... 85

5.4 Modelo de Base de Datos..... 85

5.4.1. Base de Datos para el Backend de gestión y control 86

5.4.2. Base de datos para indicadores y prototipos de modelos predictivos 90

5.5 Implementación de los indicadores clave de rendimiento (KPI) 94

5.5.1 Frontend (interfaces de usuario) 94

5.5.1.1 Estructura de la aplicación 94

5.5.1.2 Selección de colores……..... 96

5.5.1.3 Tipografía……..... 97

5.5.1.4 Iconografía……. 98

5.5.1.5 Vistas principales…….. 99

5.5.1.6 Componentes reutilizables 109

5.5.1.7 Modo Oscuro……. 119

5.5.1.8 Adaptabilidad responsiva..... 122

5.5.2 Proceso ETL (Extracción, Transformación y Carga)..... 126

5.5.3 Backend……..... 129

5.5.3.1 Backend de Gestión y Control 129

5.5.3.2 Indicadores clave de desempeño (KPI)..... 134

5.6 Prototipos de Modelos Predictivos 152

5.6.1 Obtención, Análisis y Preparación de los Datos 153

5.6.2 Entrenamiento y Evaluación de los Modelos..... 158

5.6.2.1 Criterios metodológicos para la selección del modelo..... 158

5.6.2.2 Modelo seleccionado por producto 160

5.6.2.3 Lineamientos para la elección del modelo 163

5.7 Pruebas…… 164

5.7.1 Pruebas en el frontend…. 164

5.7.1.1 Pruebas unitarias…….. 165

5.7.1.2 Pruebas de componentes 165

5.7.1.3 Resultados de las pruebas 165

5.7.2 Pruebas en el Backend…… 166

5.7.2.1 Pruebas de integración: Repositorio - Base de datos 166

5.7.2.2 Pruebas unitarias: Servicios de la capa de aplicación 167

5.7.2.3 Pruebas manuales de los endpoints 167

5.7.2.4 Resultados de las pruebas 168

6 Conclusiones 170

7 Recomendaciones Futuras 174

Referencias Bibliográficas 176

Índice de Tablas

Tabla 1 *Requerimiento funcional: Iniciar sesión* 48

Tabla 2 *Requerimiento funcional: Cerrar sesión* 49

Tabla 3 *Requerimiento funcional: Edición de cuenta* 49

Tabla 4 *Requerimiento funcional: Cambio de contraseña* 49

Tabla 5 *Requerimiento funcional: Recuperación de contraseña* 50

Tabla 6 *Requerimiento funcional: Autorización* 50

Tabla 7 *Requerimiento funcional: Autenticación* 51

Tabla 8 *Requerimiento funcional: Gestión de roles* 51

Tabla 9 *Requerimiento funcional: Gestión de usuarios* 51

Tabla 10 *Requerimiento funcional: Confirmación de cuenta* 52

Tabla 11 *Requerimiento funcional: Vista de bienvenida* 52

Tabla 12 *Requerimiento funcional: Inclusión de proceso ETL* 52

Tabla 13 *Requerimiento funcional: Cálculo y visualización de indicadores (KPI)* 53

Tabla 14 *Requerimiento funcional: Prototipo de predicción de ventas* 53

Tabla 15 *Requerimiento funcional: Persistencia de sesión iniciada* 54

Tabla 16 *Requerimiento funcional: Modo claro y oscuro* 54

Tabla 17 *Requerimiento no funcional: Rendimiento y tiempo de respuesta* 55

Tabla 18 *Requerimiento no funcional: Seguridad y privacidad de los datos* 55

Tabla 19 *Requerimiento no funcional: Escalabilidad* 56

Tabla 20 *Requerimiento no funcional: Integridad y consistencia de los datos* 56

Tabla 21 *Requerimiento no funcional: Alta disponibilidad* 56

Tabla 22 *Requerimiento no funcional: Soporte de acceso concurrente* 57

Tabla 23 *Requerimiento no funcional: Compatibilidad entre navegadores* 57

Tabla 24 *Requerimiento no funcional: Usabilidad e interfaz intuitiva* 57

Tabla 25 *Resumen entidades de la base de datos de gestión y control* 89

Tabla 26 *Endpoints del controlador de usuarios* 132

Tabla 27 *Endpoint del controlador de roles* 133

Tabla 28 *Endpoints públicos* 134

Tabla 29 *Resumen de endpoints y características asociadas* 151

Tabla 30 *Comparación de resultados SARIMA vs Prophet por producto* 160

Índice de Figuras

Figura 1 *Etapas de la metodología Box-Jenkins* 24

Figura 2 *Representación esquemática de la arquitectura limpia* 27

Figura 3 *Diagrama de casos de uso - Funcionalidades generales del software* 59

Figura 4 *Diagrama de casos de uso – Funcionalidades para usuarios autenticados* 60

Figura 5 *Diagrama de actividades para inicio de sesión* 61

Figura 6 *Diagrama de actividades para cambiar contraseña* 62

Figura 7 *Diagrama de actividades para editar información personal* 64

Figura 8 *Diagrama de actividades para la recuperación de contraseña* 65

Figura 9 *Diagrama de actividades para ver los roles* 66

Figura 10 *Diagrama de actividades para crear y editar roles* 68

Figura 11 *Diagrama de actividades para eliminar un rol* 69

Figura 12 *Diagrama de actividades para ver usuarios* 70

Figura 13 *Diagrama de actividades para crear usuario* 72

Figura 14 *Diagrama de actividades para editar usuario* 74

Figura 15 *Diagrama de actividades para eliminar usuario* 75

Figura 16 *Diagrama de actividades para entrar al panel de KPIS* 77

Figura 17 *Diagrama de actividades para ver los KPIs en un rango de fechas* 79

Figura 18 *Diagrama de la arquitectura del software* 81

Figura 19 *Diagrama base de datos gestión y control* 87

Figura 20 *Base de datos para los KPIS* 91

Figura 21 *Estructura del frontend* 95

Figura 22 *Estructura del frontend - Módulo específico* 96

Figura 23 *Paleta de colores principales de la aplicación* 97

Figura 24 *Representación visual de la fuente tipográfica Geist* 98

Figura 25 *Conjunto de iconos Lucide React empleados en la interfaz de la plataforma* 99

Figura 26 *Vista de inicio de sesión* 100

Figura 27 *Vista de recuperación de contraseña* 100

Figura 28 *Vista de ingreso de contraseña nueva* 101

Figura 29 *Vista de cuenta verificada correctamente* 101

Figura 30 *Vista de carga de verificación de cuenta* 102

Figura 31 *Vista de verificación de cuenta fallida* 102

Figura 32 *Vista de inicio* 103

Figura 33 *Vista de indicadores de desempeño* 103

Figura 34 *Vista de gestión de usuarios* 104

Figura 35 *Vista de creación de usuario* 104

Figura 36 *Vista de edición de usuarios* 105

Figura 37 *Vista de eliminación de usuario* 106

Figura 38 *Vista de gestión de roles* 107

Figura 39 *Vista de creación de rol* 107

Figura 40 *Vista de edición de rol* 108

Figura 41 *Vista de eliminación de rol* 108

Figura 42 *Sooner notificando una operación exitosa* 109

Figura 43 *Sooner notificando una operación fallida* 109

Figura 44 *Menú lateral* 110

Figura 45 *Vista de cambio de contraseña del perfil* 111

Figura 46 *Vista de edición de perfil* 112

Figura 47 *Indicador de valor único neutro* 113

Figura 48 *Indicador de valor único decreciente* 113

Figura 49 *Indicador de valor único creciente* 113

Figura 50 *Indicador de valor único en estado de carga* 114

Figura 51 *Indicador de valor único con error en su petición* 114

Figura 52 *Gráfico de línea* 115

Figura 53 *Gráfico de línea con tooltip activo* 115

Figura 54 *Gráfico de líneas múltiples* 116

Figura 55 *Gráfico de líneas múltiples con tooltip activo* 116

Figura 56 *Gráfico de columnas* 117

Figura 57 *Gráfico de columnas con tooltip activo* 117

Figura 58 *Gráfico combinado* 118

Figura 59 *Gráfico combinado con tooltip activo* 118

Figura 60 *Tabla con lista de valores* 119

Figura 61 *Vista de indicadores de desempeño en modo oscuro* 120

Figura 62 *Vista de inicio en modo oscuro* 120

Figura 63 *Vista de gestión de usuarios en modo oscuro* 121

Figura 64 *Vista de gestión de roles en modo oscuro* 121

Figura 65 *Vista de inicio de sesión en dispositivos móviles* 122

Figura 66 *Menú lateral en dispositivos móviles* 123

Figura 67 *Vista de inicio en dispositivos móviles* 124

Figura 68 *Vista de gestión de usuarios en dispositivos móviles* 125

Figura 69 *Conexiones entre Bases de Datos* 127

Figura 70 *Proceso de Transformación y Carga.* 128

Figura 71 *Estructura del proyecto en Visual Studio* 130

Figura 72 *Endpoints KPIS* 134

Figura 73 *KPI Total Ventas* 136

Figura 74 *KPI Precio Total Ventas* 137

Figura 75 *KPI Utilidad Total Ventas* 138

Figura 76 *KPI Margen de Ganancia* 139

Figura 77 *KPI Evolución Ingresos Utilidad Ventas* 140

Figura 78 *KPI Evolución Ventas Mensuales* 141

Figura 79 *KPI Ticket Promedio Mensual* 142

Figura 80 *KPI Ventas Totales Anuales* 143

Figura 81 *Ventas por Sucursal* 144

Figura 82 *KPI de Ventas Trimestrales* 145

Figura 83 *KPI de Evolución de Ventas Anuales* 146

Figura 84 *KPI de Evolución de Ingresos Totales por Ventas Mensuales* 147

Figura 85 *KPI de Evolución del Margen de Ganancia* 148

Figura 86 *KPI de Evolución del Ticket Promedio* 149

Figura 87 *KPI de Productos Más Vendidos* 150

Figura 88 *KPI de Productos Menos Vendidos* 151

Figura 89 *Ventas diarias del producto A* 154

Figura 90 *Ventas diarias del producto B* 155

Figura 91 *Ventas diarias del producto C* 155

Figura 92 *Ventas diarias del producto A (Datos ordenados)* 156

Figura 93 *Ventas diarias del producto B (Datos ordenados)* 157

Figura 94 *Ventas diarias del producto C (Datos ordenados)* 157

Figura 95 *Predicción Sarima Producto A* 161

Figura 96 *Predicción Prophet Producto B* 162

Figura 97 *Predicción Prophet Producto C* 162

Figura 98 Resultados de pruebas de frontend 166

Figura 99 *Resultados de las pruebas del backend* 169

Resumen

Título: Desarrollo de una plataforma web para el análisis estadístico y el apoyo a la toma de decisiones empresariales en el sector retail colombiano

Autor: Daniel Alejandro León Ortiz, Jhon Danilo Rincón Maldonado y Yefferson Olivos Rodríguez.

Palabras Clave: inteligencia de negocios, toma de decisiones, visualización interactiva, ventas, ARIMA, prophet, indicadores de desempeño, plataforma web, sector retail.

Descripción: El presente documento presenta el desarrollo de una plataforma web diseñada para apoyar la toma de decisiones estratégicas en empresas del sector retail colombiano, mediante el análisis estadístico y la visualización de información clave. El software permite consultar indicadores de desempeño (KPIs) y analizar el comportamiento histórico de ventas que facilite el reconocimiento de tendencias. Se diseñó e implementó un proceso de extracción, transformación y carga (ETL) que adapta los datos provenientes de una base central a una estructura alineada con los requerimientos del software. La arquitectura del software fue construida con enfoque escalable y multicliente, bajo los principios de arquitectura limpia, para que múltiples organizaciones trabajen de forma segura e independiente dentro de una misma solución. Además, se incorporó un módulo de auditoría que registra las acciones de los usuarios, fortaleciendo la trazabilidad y el control dentro del software. Adicionalmente se incluyen modelos de pronóstico basado en series temporales (ARIMA, SARIMA, Prophet), que contribuye con la estimación la demanda futura y apoyar la planificación comercial. Adicionalmente, se presentan las pruebas realizadas de validación enfocadas en el rendimiento, la funcionalidad y la integridad de los datos que aporte estabilidad y confiabilidad al software. El desarrollo es un aporte a la transformación digital de las empresas, una herramienta accesible, adaptable y basada en el aprovechamiento estratégico de los datos.

---

* Trabajo de grado

** Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Ingeniería de sistemas. Director: Urbano Eliécer Gómez Prada. Doctor en Tecnología Educativa

Abstract

Title: Development of a Web Platform for Statistical Analysis and Decision-Making Support in the Colombian Retail Sector

Author(s): Daniel Alejandro León Ortiz, Jhon Danilo Rincón Maldonado, and Yefferson Olivos Rodriguez

Key Words: business intelligence, decision-making, interactive dashboards, sales, ARIMA, prophet, performance indicators, web platform, retail sector.

Description: This document presents the development of a web platform designed to support strategic decision-making in Colombian retail companies through statistical analysis and visualization of key information. The software enables users to consult performance indicators (KPIs) and analyze historical sales behavior to identify trends. An extraction, transformation, and loading (ETL) process were designed and implemented to adapt data from a central source into a structure aligned with the software's requirements. The system architecture was built with a scalable, multi-tenant approach, following clean architecture principles to ensure secure and independent use by multiple organizations within the same solution. Additionally, an auditing module was integrated to record user actions, strengthening traceability and control within the platform. The system also incorporates forecasting models based on time series (ARIMA, SARIMA, Prophet) to estimate future demand and support business planning. Validation tests were conducted, focusing on performance, functionality, and data integrity to ensure the software's stability and reliability. This development contributes to the digital transformation of businesses, offering an accessible, adaptable tool grounded in the strategic use of data.

---

* Undergraduate Thesis

**Faculty of Physicomechanical Engineering. School of Systems and Computer Engineering. Systems Engineering. Advisor: Urbano Eliécer Gómez Prada. Systems Engineer, Doctor in Educational Technology

Introducción

El análisis de datos, conocido como "data analytics", ha transformado la manera en que las organizaciones transforman grandes volúmenes de información en conocimiento estratégico, ya que permite identificar patrones, comprender las tendencias del mercado y optimizar decisiones empresariales, aspectos cruciales en un entorno cada vez más competitivo. De acuerdo con esto, Predik Data-Driven (2024) señala que la integración de inteligencia de mercados con un modelo de datos sólido permite a las organizaciones obtener una visión estratégica que mejora la efectividad de sus decisiones. En un contexto global impulsado por los datos, el análisis avanzado deja de ser una simple herramienta para convertirse en un elemento clave del éxito organizacional.

El análisis de datos permite a las empresas obtener una visión detallada y profunda de sus operaciones, productos y clientes, facilitando así la toma de decisiones estratégicas. Al convertir grandes volúmenes de datos en información valiosa, las organizaciones pueden identificar patrones y tendencias, anticipar necesidades y mejorar la eficiencia operativa. Además, al optimizar procesos internos, se logra personalizar la experiencia del cliente y genera mayor satisfacción y fidelidad. Finalmente, al transformar datos crudos en conocimiento accionable, las empresas tienen la capacidad de adaptarse rápidamente a los cambios del mercado y mantenerse competitivas (AWS, 2022a).

El análisis de datos para convertir en información los grandes volúmenes de datos en las organizaciones se pueden enfocar al sector retail colombiano para mejorar los procesos de comercialización, por ello el propósito de esta práctica empresarial, busca desarrollar un sistema que realice analítica de datos.

Durante la práctica empresarial, los estudiantes se enfocarán en desarrollar una plataforma web centrada en el análisis de datos, con el objetivo de visualizar indicadores clave de desempeño

(KPIs) y crear un prototipo predictivo de ventas. Este proyecto busca crear un software robusto que no solo facilite la toma de decisiones estratégicas a través de la visualización de métricas, sino que también permita predecir tendencias futuras basadas en datos históricos. Como señala el tutor del proyecto, Juan Félix Ballesteros Suárez, “el software debe ser flexible, escalable y capaz de adaptarse a las futuras necesidades de la empresa, asegurando que el uso de los datos se maximice para mejorar la toma de decisiones” (J. F. Ballesteros Suárez, comunicación personal, 16 de octubre de 2024).

1 Planteamiento y Justificación del Problema

Muchas empresas enfrentan dificultades para gestionar y aprovechar eficazmente los grandes volúmenes de datos que generan a diario, lo que obstaculiza su transformación digital. De acuerdo con un estudio de Dell Technologies, el 95% de las organizaciones en Latinoamérica lidian con la gestión de datos y casi dos tercios tienen problemas para extraer información útil. Esta situación afecta su capacidad de tomar decisiones informadas, impactando directamente en su competitividad y subrayando la necesidad de herramientas especializadas y plataformas avanzadas para el análisis de datos (Goncalves, 2024).

Uno de los principales factores que contribuyen a esta problemática es la carencia de herramientas especializadas para centralizar, procesar y visualizar la información de forma eficiente. En respuesta, el Proyecto FÉNIX, actualmente en sus etapas iniciales dentro de la empresa APL, busca desarrollar un conjunto integral de soluciones tecnológicas, como sistemas de gestión CRM y POS, que aún no han sido implementados. Sin embargo, la falta de una plataforma específica para el análisis avanzado de datos en esta fase inicial limita la capacidad de la empresa para optimizar el uso de la información generada, obstaculizando la identificación de patrones y tendencias que son esenciales para tomar decisiones estratégicas (J. F. Ballesteros Suárez, comunicación personal, 16 de octubre de 2024).

Las consecuencias de no contar con un software adecuado para el análisis de datos son profundas. La empresa se ve obligada a tomar decisiones sin una base sólida de información, lo que puede llevar a resultados erróneos y afectar la rentabilidad. Además, sin un análisis adecuado, se pierden oportunidades clave para mejorar la eficiencia operativa y responder rápidamente a los cambios del mercado. A largo plazo, esto puede generar una disminución en la competitividad de

la empresa, ya que se ve incapaz de detectar patrones importantes o de adaptarse con rapidez a un entorno empresarial cada vez más dinámico (Datamind, 2023).

En consecuencia, se justifica el desarrollo de una plataforma web para el Análisis Estadístico y Optimización de Ventas en el Proyecto FÉNIX, que permita a la empresa APL centralizar y procesar eficientemente la información generada. Esta herramienta no solo mejorará la precisión en la toma de decisiones, sino que también optimizará los procesos internos y permitirá identificar oportunidades de negocio, fortaleciendo así la posición competitiva de la empresa en el futuro.

2 Objetivos

2.1 Objetivo General

Desarrollar un software web comercial con visualización interactiva y pronóstico de ventas mediante una arquitectura escalable para el apoyo en la toma de decisiones estratégicas de la organización.

2.2 Objetivos Específicos

1. Definir los requerimientos funcionales y no funcionales del software, asegurando que cubran las necesidades del usuario y los objetivos del proyecto.
2. Modelar los casos de uso y los diagramas de actividades del software para representar los flujos de trabajo y las interacciones clave.
3. Definir una arquitectura de software que gestione múltiples clientes dentro de una misma aplicación, permitiendo la correcta separación y seguridad de los datos.
4. Diseñar la estructura de la base de datos que soportará el software, procurando eficiencia y consistencia en el manejo de la información.
5. Definir e implementar indicadores de desempeño (KPIs) enfocados en medir aspectos de ventas, para monitorear y evaluar estadísticas clave que permitan analizar el impacto de las estrategias empresariales.
6. Desarrollar un prototipo de modelo predictivo basado en series temporales, para analizar datos históricos de ventas y estimar la demanda futura, optimizando así la toma de decisiones estratégicas en la planificación comercial.
7. Realizar pruebas para la validación del rendimiento, la funcionalidad y la seguridad de los datos del software antes de su entrega final.

3 Marco de referencia

3.1 Marco contextual

Con el fin de facilitar la comprensión del presente proyecto, se presentan y explican a continuación los conceptos más relevantes.

3.1.1 Indicadores Clave de Desempeño (KPIs)

Los Indicadores Clave de Desempeño (KPIs) son métricas cuantitativas que permiten evaluar el progreso hacia objetivos definidos y facilitan la toma de decisiones basadas en información real, en lugar de percepciones subjetivas (Xairó, 2023). En las organizaciones, estos indicadores son fundamentales para monitorear la eficiencia de los procesos, la efectividad de las estrategias y el nivel de cumplimiento de las metas planteadas.

De acuerdo con SYDLE (2023), los KPIs pueden clasificarse en distintas categorías, como financieros, de productividad, de clientes o estratégicos, entre otros. En este proyecto, los KPIs fueron considerados como un concepto clave para entender la importancia de medir y gestionar el desempeño empresarial, dado que aportan información estructurada y objetiva que puede apoyar la toma de decisiones en distintos niveles de la organización.

3.1.2 Modelos Predictivos

Los modelos predictivos son herramientas analíticas que permiten identificar patrones en datos históricos y generar aproximaciones útiles para anticipar posibles comportamientos futuros (Probabilidad y Estadística, 2024). En este proyecto no se implementaron como soluciones definitivas, sino como prototipos exploratorios cuyo propósito fue apoyar la gestión de inventarios

y servir como insumo en la toma de decisiones estratégicas dentro del sector retail. Para ello se analizaron tres enfoques principales: ARIMA, SARIMA y Prophet.

3.1.2.1 ARIMA

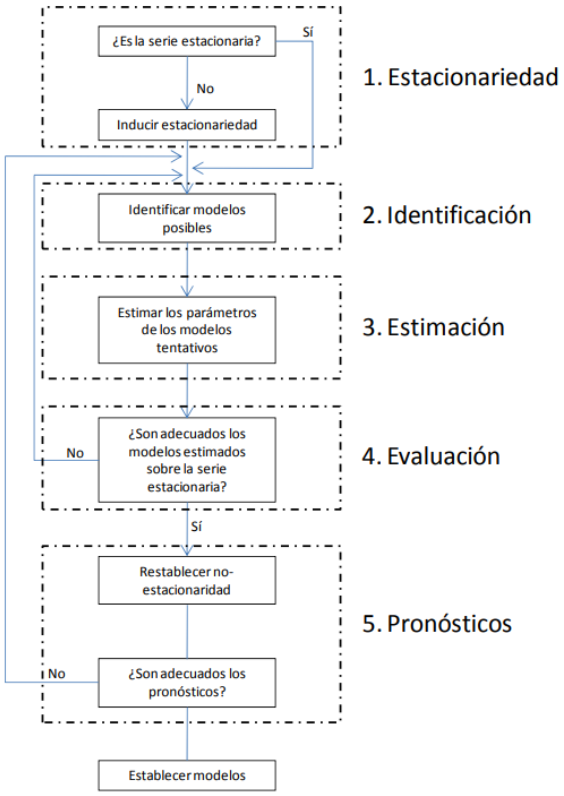
El modelo ARIMA (AutoRegressive Integrated Moving Average) ha sido ampliamente aplicado en contextos financieros, industriales y de gestión de inventarios debido a su capacidad para representar relaciones temporales en series univariantes (Saadeddin, 2024; UNAM, 2013). Este modelo combina tres componentes: autorregresión (AR), integración mediante diferenciación (I) y media móvil (MA), lo que le permite trabajar con datos históricos sin necesidad de variables externas (Probabilidad y Estadística, 2024).

La construcción de modelos ARIMA se realiza bajo la metodología Box-Jenkins, que establece un proceso sistemático de cinco etapas: verificación de estacionariedad, identificación de parámetros, estimación de coeficientes, validación de residuos y pronóstico (UNAM, 2013). Dichas etapas se resumen en el esquema mostrado en la Figura 1, el cual ilustra gráficamente el flujo metodológico aplicado para este tipo de modelos.

En este proyecto, ARIMA fue empleado como punto de partida por su fundamentación estadística y por la existencia de procedimientos estandarizados para su implementación. Aunque exige que la serie sea estacionaria, la metodología Box-Jenkins ofrece técnicas como la diferenciación para lograr esta condición. De esta manera, ARIMA permitió sentar las bases del análisis y comprender la dinámica general de las series.

Figura 1

Etapas de la metodología Box-Jenkins



Nota. Tomado de *Modelación ARIMA,* por Jenkins, 2013,

http://www.ptolomeo.unam.mx:8080/jspui/bitstream/132.248.52.100/363/7/A7.pdf

3.1.2.2 Sarima

El modelo SARIMA (Seasonal ARIMA) amplía las capacidades de ARIMA al incorporar un componente estacional explícito, representado en su notación como SARIMA(p,d,q)(P,D,Q)_s. Esta estructura lo hace especialmente útil en series donde los patrones estacionales se repiten a intervalos regulares, como la demanda energética, la producción agrícola o las ventas periódicas (Neptune.ai, 2024).

En este proyecto, SARIMA se consideró como un prototipo exploratorio para abordar series en las que se identificaron ciclos de ventas recurrentes. Su incorporación permitió contrastar los resultados obtenidos con ARIMA y valorar cómo la estacionalidad explícita mejora la representación de las series temporales. De esta forma, SARIMA ofreció una perspectiva complementaria para el análisis.

3.1.2.3 Prophet

El modelo Prophet, desarrollado por Meta, descompone una serie temporal en tres componentes principales: tendencia, estacionalidad y eventos especiales, como días festivos (Taylor & Letham, 2017). A diferencia de ARIMA y SARIMA, Prophet no requiere que la serie sea estacionaria, lo que facilita su aplicación en datos con irregularidades, valores atípicos o cambios estructurales.

En el contexto de este proyecto, Prophet se aplicó como una alternativa práctica y flexible, particularmente útil en escenarios donde se manejan múltiples productos y se requiere un ajuste ágil sin configuración compleja de parámetros. Su facilidad para incorporar calendarios de festivos nacionales también permitió valorar su utilidad en contextos como el retail colombiano, donde las empresas manejan un número elevado de productos y no siempre es viable realizar ajustes manuales de parámetros en cada caso.

3.1.3 Arquitectura multicliente

La arquitectura multicliente (multi-tenant) es un enfoque de diseño que permite que una sola aplicación dé servicio a múltiples clientes independientes o tenants. Aunque estos comparten recursos de infraestructura y componentes comunes, cada cliente mantiene un entorno lógico aislado, lo que garantiza privacidad y seguridad en sus datos (Bautista et al., 2024).

Este enfoque resulta relevante en el contexto del proyecto porque el sistema de gestión de usuarios, roles y permisos debía concebirse como una solución que pudiera ser usada por distintas empresas, cada una con independencia en sus datos pero compartiendo la misma plataforma tecnológica.

Dentro de esta arquitectura existen tres posibles estrategias para la persistencia de datos (Alonso, 2018):

1. Base de datos independiente por cliente, con máximo aislamiento pero mayor costo de mantenimiento.
2. Esquemas separados dentro de una misma base de datos, que equilibran aislamiento y eficiencia.
3. Esquema compartido con identificadores por cliente, el más eficiente en recursos pero con mayores desafíos de seguridad.

La elección de un enfoque multicliente permite visualizar cómo el sistema podría escalar a futuro y facilitar la administración centralizada, al mismo tiempo que mantiene flexibilidad en el manejo de datos según las necesidades de cada empresa.

3.1.4 Arquitectura limpia

La arquitectura limpia, propuesta por Robert C. Martin, busca que la lógica de negocio sea independiente de detalles técnicos como bases de datos, frameworks o interfaces gráficas (Calderón, 2021). Su organización concéntrica establece capas donde las reglas de negocio permanecen en el núcleo, mientras que las tecnologías específicas ocupan la periferia. La regla de dependencia garantiza que el flujo siempre vaya de lo externo hacia lo interno, nunca al contrario (Arias et al., 2021).

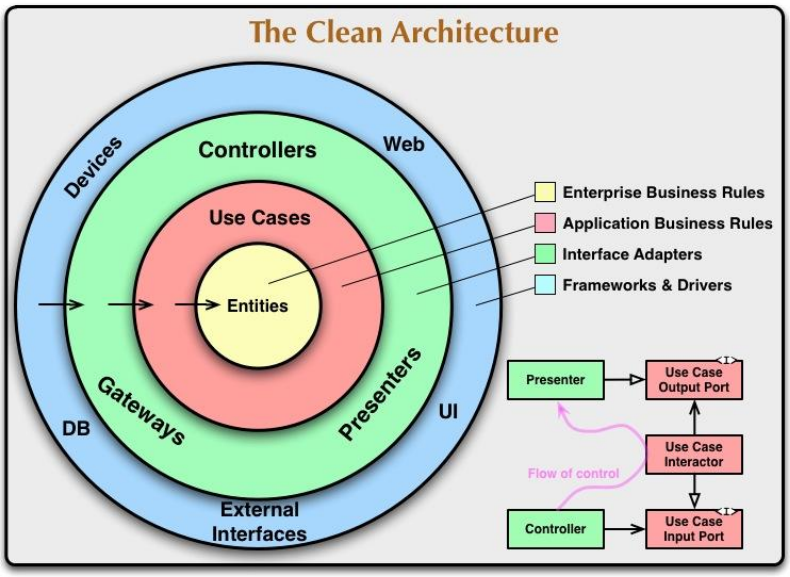
Como se muestra en la Figura 1, las capas incluyen:

1. Entidades del dominio, que encapsulan las reglas más generales.
2. Casos de uso, que aplican esas reglas en situaciones concretas.
3. Interfaces de entrada, que definen la interacción con usuarios o sistemas externos.
4. Frameworks e infraestructura, donde residen las tecnologías específicas.

Este enfoque fue adoptado en el proyecto porque facilita la mantenibilidad del sistema, permitiendo cambiar tecnologías (por ejemplo, el motor de base de datos) sin alterar la lógica de negocio. Además, promueve la escalabilidad y favorece la realización de pruebas unitarias al mantener un alto nivel de desacoplamiento (Cambarieri et al., 2020).

Figura 2

Representación esquemática de la arquitectura limpia



Nota. Tomado de *The Clean Architecture,* por R.C. Martin, 2012, https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

3.1.5 Proceso ETL (Extracción, Transformación y Carga)

El proceso ETL (Extract, Transform and Load) es una práctica fundamental en la gestión de datos, ya que permite extraer información desde distintas fuentes, transformarla para cumplir requerimientos analíticos y cargarla en un sistema de destino unificado (AWS, 2022b). Según Oracle (2022), este proceso funciona como un puente entre los sistemas operacionales y los analíticos, consolidando información dispersa en estructuras que facilitan su análisis.

En el proyecto se implementó un proceso ETL que permitió traer información relevante desde una base de datos externa hacia la base de datos del sistema. Este procedimiento aseguró que los datos quedaran estandarizados y listos para ser utilizados en los prototipos de análisis, como el cálculo de indicadores de gestión y la preparación de las series temporales para el modelado. La lógica del proceso incluyó:

1. Extracción: recuperación de datos desde la base de datos de origen.
2. Transformación: limpieza, depuración y adecuación de la información para garantizar consistencia.
3. Carga: inserción de los datos procesados en la base de datos del proyecto, para su posterior consulta y uso en el sistema.

De acuerdo con Microsoft (2018), un buen diseño ETL debe considerar la calidad de las fuentes, el volumen de información y la frecuencia de actualización. Estos criterios fueron aplicados de manera proporcional a los requerimientos del proyecto, asegurando que la información extraída y transformada cumpliera con condiciones básicas de integridad y estuviera disponible para apoyar la toma de decisiones.

3.2 Marco tecnológico

A continuación, se detallan las tecnologías empleadas durante el desarrollo del proyecto.

3.2.1 Ecosistema tecnológico del frontend

En este apartado se abordarán los lenguajes y tecnologías que definen la estructura y el diseño de la página web desarrollada para el proyecto.

3.2.1.1 Frameworks y bibliotecas de interfaz de usuario

En el desarrollo del proyecto, React desempeñó un papel central como biblioteca de JavaScript creada por Meta, ya que permite la construcción de interfaces de usuario mediante un sistema de componentes reutilizables, lo que facilita la organización del código y reduce los errores en el proceso de renderizado (MDN, 2025). A diferencia de otros frameworks, React no impone convenciones rígidas sobre la organización del código, lo que posibilitó adaptar la estructura del proyecto a sus propias necesidades (MDN, 2025). Aunque no incluye todas las funcionalidades de otras alternativas, su ecosistema ofrece suficientes herramientas y la posibilidad de integrar librerías externas, lo que permitió incorporar únicamente los recursos necesarios para cumplir los objetivos planteados.

En estrecha relación con ello, la navegación dentro de la aplicación se gestionó mediante React Router, una biblioteca que facilita el manejo de rutas dinámicas en aplicaciones de una sola página, vinculando URLs con componentes específicos y ofreciendo una experiencia fluida sin recargas completas (React Router, s.f.). Su adopción respondió a la flexibilidad que brinda a través de distintos modos de configuración, capaces de adaptarse tanto a escenarios simples como a casos

que requieren un mayor control sobre el flujo de datos y la estructura general de la aplicación (React Router, s.f.).

De manera complementaria, para la construcción de interfaces consistentes se incorporó Shadcn/ui, un sistema de componentes de código abierto que proporciona elementos accesibles y altamente personalizables, entregando además el código fuente completo para su adaptación (Shadcn, s.f.). Esta característica resultó clave, ya que permitió integrar componentes coherentes directamente en la aplicación y, al mismo tiempo, garantizar un diseño flexible. Asimismo, su compatibilidad con herramientas de inteligencia artificial y la disponibilidad de estilos predeterminados optimizaron tanto la implementación como la personalización de la interfaz (Shadcn, s.f.).

Finalmente, en lo que respecta al diseño visual, se empleó Tailwind CSS, un framework desarrollado por Tailwind Labs Inc. que ofrece clases utilitarias prediseñadas para crear interfaces personalizadas sin necesidad de archivos CSS extensos, lo que favorece la consistencia y eficiencia en el desarrollo (GeeksforGeeks, 2024). Gracias a su integración, fue posible agilizar la construcción de estilos responsivos y reducir la complejidad derivada de la gestión de hojas de estilo tradicionales, lo que repercutió positivamente en la mantenibilidad del proyecto y en la optimización del tiempo de desarrollo.

3.2.1.2 Lenguajes y tipado estático

Como lenguaje de programación principal se empleó TypeScript, desarrollado por Microsoft, el cual añade un sistema de tipado a JavaScript que facilita la detección anticipada de errores y mejora la confiabilidad del desarrollo (Microsoft, s.f.a). Frente a las limitaciones del tipado dinámico de JavaScript, este lenguaje permitió aumentar la legibilidad y organización del código, favoreciendo tanto el trabajo de los desarrolladores actuales como la futura incorporación

de nuevos integrantes. Además, su compatibilidad con React lo convirtió en una herramienta sólida para el desarrollo de las interfaces implementadas en el proyecto.

3.2.1.3 Construcción y gestión de dependencias

En cuanto a la construcción y gestión de dependencias, se optó por herramientas que aportaran tanto eficiencia como simplicidad al flujo de trabajo. En este sentido, Pnpm (Performant NPM) se implementó como gestor de paquetes de código abierto, ya que optimiza la instalación y administración de dependencias mediante un almacén global basado en direcciones de contenido, lo que posibilita compartir librerías entre proyectos a través de enlaces duros (Refine, 2024). Esta característica resultó especialmente valiosa en el proyecto, dado que redujo significativamente el uso de espacio en disco y los tiempos de instalación, favoreciendo un desarrollo más ágil.

De manera complementaria, se empleó Vite, una herramienta de desarrollo que aprovecha los módulos ES nativos del navegador para acelerar la recarga de cambios gracias al mecanismo de Hot Module Replacement (HMR), lo que incrementó la eficiencia en la etapa de construcción de la aplicación (VoidZero, s.f.). Además, en el proceso de compilación para producción recurre a Rollup para generar paquetes optimizados con divisiones de código inteligentes, ofreciendo también compatibilidad con frameworks modernos como React. Estas características lo convirtieron en un recurso esencial dentro de la arquitectura tecnológica del proyecto (VoidZero, s.f.).

3.2.1.4 Comunicación con APIs y servicios externos

En cuanto a la comunicación con APIs y servicios externos, se empleó Axios, una biblioteca cliente HTTP basada en promesas que permite realizar solicitudes desde el navegador o Node.js, gracias a su carácter isomórfico (Axios, s.f.).

En este proyecto se utilizó por su sintaxis sencilla y por automatizar procesos frecuentes en la comunicación con APIs, como el manejo de interceptores o la cancelación de solicitudes, lo que resultó más adecuado que recurrir a alternativas nativas o a bibliotecas más complejas.

3.2.1.5 Calidad y estilo del código

En cuanto al aseguramiento de la calidad y la estandarización del estilo, se implementaron herramientas que permitieron mantener un código consistente, legible y libre de errores comunes. Por un lado, ESLint se utilizó como herramienta de análisis estático para identificar errores y verificar el cumplimiento de buenas prácticas en JavaScript mediante reglas configurables y ampliables con plugins (ESLint, s.f.). Su aplicación en el proyecto garantizó la consistencia del código y la prevención de fallos potenciales, además de aprovechar funciones como la corrección automática y la integración con el entorno de desarrollo, lo que fortaleció la calidad y mantenibilidad del software.

Por otro lado, Prettier se incorporó para automatizar el formateo del código, aplicando un estilo uniforme que favoreció la legibilidad y la integración de nuevos desarrolladores, además de optimizar tiempos al eliminar discusiones sobre convenciones de estilo (Prettier, s.f.). En conjunto con ESLint, cumplió un rol complementario: mientras esta última se enfocó en la calidad y detección de errores, Prettier garantizó un formato limpio y consistente en toda la base de código.

3.2.1.6 Pruebas y aseguramiento de calidad

Dentro del aseguramiento de la calidad del software, se empleó Vitest, un framework de pruebas moderno que se integra de manera nativa con Vite y reutiliza su configuración y plugins, garantizando coherencia entre los entornos de desarrollo y de pruebas (Allotey, 2025). Su adopción en el proyecto se debió a su alto rendimiento, ya que ejecuta pruebas en paralelo y ofrece

retroalimentación inmediata mediante su modo de observación. Asimismo, su compatibilidad con TypeScript, JSX y la API de Jest facilitó una experiencia de validación ágil y eficiente (Vitest, s.f.).

Complementariamente, se utilizó Testing Library, un conjunto de herramientas que promueve la escritura de pruebas centradas en la experiencia del usuario, al fomentar interacciones con la aplicación de una forma similar a la que tendría un usuario real (Testing Library, 2021). Esta característica permitió desarrollar pruebas más robustas frente a cambios internos de implementación y alineadas con el comportamiento observable de la interfaz. Además, su integración con frameworks modernos y entornos de prueba como Jest aseguró la confiabilidad y mantenibilidad del software desarrollado (Testing Library, 2021).

3.2.2 Ecosistema tecnológico del backend

En el desarrollo del backend de la plataforma web se empleó un conjunto de tecnologías que, en conjunto, permitieron implementar una solución escalable, segura y alineada con la arquitectura limpia definida para el proyecto. Estas herramientas abarcan desde el lenguaje de programación y la plataforma de desarrollo, hasta la gestión de datos y los mecanismos de seguridad y calidad del software.

3.2.2.1 Lenguaje y plataforma de desarrollo

El backend fue implementado utilizando C# dentro del ecosistema .NET, lo cual garantizó un entorno robusto y flexible para la construcción de la plataforma. C# es un lenguaje moderno, orientado a objetos y de tipado fuerte, diseñado por Microsoft para crear aplicaciones versátiles y seguras (Microsoft, 2022; TRBL Services, 2021). Su estrecha integración con .NET facilitó la

adopción de principios de arquitectura limpia, favoreciendo la mantenibilidad del software y la posibilidad de desacoplar la lógica del negocio de los detalles técnicos (Besoftware, 2020).

Por su parte, .NET es una plataforma de desarrollo de código abierto y multiplataforma, que permite crear aplicaciones para distintos entornos, desde soluciones web hasta servicios en la nube. Gracias a su entorno de ejecución común (CLR), distintos lenguajes pueden interoperar de forma coherente, lo que aporta versatilidad al ecosistema (Microsoft, 2022b). En este proyecto, .NET brindó acceso a bibliotecas y herramientas que facilitaron la implementación de la lógica de negocio, el acceso a datos y la integración con servicios externos, asegurando escalabilidad y sostenibilidad en el tiempo.

3.2.2.2 Gestión y acceso a datos

Para la gestión de la información se utilizó SQL Server como sistema de gestión de bases de datos relacional (RDBMS). Este motor permitió almacenar y recuperar datos de forma eficiente, garantizando integridad, seguridad y disponibilidad en el manejo de la información (Microsoft, 2025a). Además, ofreció soporte para consultas complejas y administración de roles de acceso, aspectos fundamentales en el sector retail donde la consistencia y seguridad de los datos resultan críticas (Actian, 2022).

El acceso a esta base de datos se realizó mediante Dapper, una librería ORM ligera para .NET, que facilitó la interacción con SQL Server sin sacrificar rendimiento. A diferencia de otros ORM más complejos, Dapper permitió mapear consultas SQL directamente a objetos en C#, ofreciendo simplicidad y velocidad en el acceso a datos (Learn Dapper, 2024). La combinación de SQL Server con Dapper resultó adecuada para equilibrar eficiencia en las operaciones y flexibilidad en la manipulación de la información.

3.2.2.3 Seguridad y calidad del software

La seguridad y la calidad fueron abordadas mediante un conjunto de tecnologías complementarias. En primer lugar, la autenticación con JSON Web Tokens (JWT) permitió establecer un mecanismo de acceso seguro y escalable, en el que los usuarios podían autenticarse mediante tokens firmados digitalmente que garantizan la integridad de la información transmitida (JWT, 2015; 1Kosmos, 2023).

Para reforzar la protección de credenciales se empleó BCrypt, un algoritmo de hashing diseñado para proteger contraseñas mediante salting y un factor de complejidad configurable, lo que dificulta los ataques de fuerza bruta (Code Maze, 2024).

En cuanto a la calidad del software, se implementaron pruebas unitarias con xUnit, un framework diseñado para .NET que facilita la escritura de pruebas limpias, organizadas y ejecutables en paralelo (xUnit.net, 2019; Vinugayathri, 2019). Estas pruebas fueron complementadas con el uso de Moq, un framework de mocking que permitió simular dependencias externas, favoreciendo la aislación de los componentes y la verificación precisa de su comportamiento (Kalinda, 2023; Code Maze, 2023).

De esta manera, la combinación de JWT, BCrypt, xUnit y Moq permitió garantizar tanto la seguridad de la plataforma como la fiabilidad de su comportamiento bajo distintos escenarios de prueba.

3.2.3 Tecnologías para análisis de datos y modelos estadísticos

Además del ecosistema tecnológico del backend principal, el proyecto incorporó un conjunto de herramientas en Python orientadas al análisis de datos y a la construcción de prototipos estadísticos. Estas tecnologías facilitaron la preparación de la información, la estructuración de

indicadores de desempeño y la experimentación con modelos analíticos como apoyo a la gestión de inventarios y ventas.

En primer lugar, se utilizaron NumPy y Pandas, dos bibliotecas ampliamente reconocidas en el ecosistema científico de Python. NumPy proporciona estructuras como arreglos multidimensionales y operaciones matemáticas optimizadas (Numpy, 2022), mientras que Pandas ofrece estructuras de datos como DataFrame y Series, que permiten manipular información tabular de manera flexible (Datascientest, 2024). En el proyecto, estas bibliotecas fueron la base para organizar y transformar los datos que luego se emplearon en las siguientes fases.

Posteriormente, se usaron Statsmodels y pmdarima, dos bibliotecas enfocadas en el análisis de series temporales. Statsmodels incluye implementaciones de modelos estadísticos como ARIMA y SARIMA, así como funciones de validación y análisis de resultados (Statsmodels, 2024). Por su parte, pmdarima simplifica la selección automática de parámetros mediante la función auto_arima (Smith y Smith, 2023), lo que permitió explorar diferentes configuraciones de modelos de manera más ágil.

Asimismo, se incorporó la librería Prophet, desarrollada por Meta, que implementa un enfoque aditivo basado en componentes de tendencia, estacionalidad y eventos especiales (Meta, 2025). Su diseño sencillo y flexible la hizo útil para generar prototipos adicionales y contrastar sus salidas con las de los modelos clásicos de series temporales.

Finalmente, para conectar este componente analítico con la plataforma web, se utilizó FastAPI, un framework que facilita la creación de servicios RESTful en Python, incluyendo validación automática de datos y documentación interactiva (FastAPI, s.f.). De este modo, los resultados generados en Python pudieron integrarse con el resto de las funcionalidades de la plataforma.

3.2.4 Herramientas Complementarias de Desarrollo

En esta sección se presentarán las herramientas adicionales utilizadas durante el desarrollo del proyecto, las cuales fueron fundamentales para optimizar el flujo de trabajo, facilitar la implementación y mejorar la eficiencia del desarrollo.

3.2.4.1 Cursor

Cursor es un editor de código impulsado por inteligencia artificial, desarrollado por Anysphere y basado en Visual Studio Code, lo que asegura compatibilidad con extensiones y configuraciones existentes (Datacamp, 2024).

En este proyecto se valoró por sus funciones avanzadas de IA, como la generación y reescritura inteligente de código, el autocompletado predictivo y la capacidad de realizar consultas en lenguaje natural sobre la base de código, características que optimizaron la productividad y facilitaron el desarrollo (Cursor, s.f.).

3.2.4.2 Azure devops

Azure DevOps es un conjunto de servicios en la nube que facilita la planificación de proyectos, la colaboración en el desarrollo y la entrega continua de software mediante pipelines automatizados (Microsoft, s.f.b). En este proyecto se utilizó para gestionar el ciclo de vida del desarrollo, permitiendo optimizar la organización del trabajo y aumentar la eficiencia en la implementación.

3.2.4.3 Visual Studio

Visual Studio es un entorno de desarrollo integrado (IDE) creado por Microsoft, diseñado para facilitar el desarrollo de aplicaciones en múltiples lenguajes, incluyendo C#, C++ y Python, entre otros. Esta herramienta ofrece características avanzadas como depuración, administración de versiones, diseño visual de interfaces y soporte para pruebas automatizadas, lo que la convierte en una solución integral para desarrolladores que buscan eficiencia y robustez en sus proyectos (Microsoft, 2022c).

3.2.4.4 SQL Server Management Studio 20

SQL Server Management Studio es una herramienta integral de administración para SQL Server, desarrollada por Microsoft. Proporciona una interfaz gráfica que permite configurar, administrar y supervisar instancias del motor de base de datos de SQL Server, así como ejecutar consultas y scripts de manera eficiente. SSMS está diseñado para facilitar tanto tareas administrativas como de desarrollo, ofreciendo un entorno robusto para la gestión de bases de datos relacionales (Microsoft, 2025b).

3.2.4.5 Postman

Postman es una herramienta que facilita el desarrollo, prueba y documentación de APIs mediante una interfaz que permite enviar solicitudes HTTP, analizar respuestas y automatizar pruebas, optimizando así la construcción de servicios web (Postman, 2019).

Además, en este proyecto se aprovechó su capacidad de trabajo colaborativo, como el uso compartido de colecciones y la generación automática de documentación, lo que lo consolidó como un recurso clave en la implementación de arquitecturas REST (Muradas, 2019).

3.3 Antecedentes del tema

3.3.1 Google Analytics

Google Analytics es una herramienta líder en el análisis de tráfico web, proporcionando a las empresas información detallada sobre el comportamiento de los usuarios en sus sitios, como visitantes, conversiones y duración de sesiones. Su capacidad para segmentar audiencias y generar informes detallados permite identificar patrones clave en el uso de plataformas digitales (Google, s.f.).

Aunque su enfoque es el análisis web, su capacidad para gestionar grandes volúmenes de datos y generar visualizaciones comprensibles sirve como un modelo a seguir para el desarrollo del proyecto, especialmente en cuanto a la creación de dashboards y la visualización de tendencias.

3.3.2 Power BI

Power BI es una plataforma de inteligencia empresarial que conecta múltiples fuentes de datos y ofrece análisis interactivos mediante cuadros de mando personalizados y análisis predictivos. Las empresas lo utilizan para visualizar el rendimiento operativo y comercial, permitiendo una toma de decisiones más informada (Microsoft, s.f.c).

Para el proyecto, Power BI representa un excelente ejemplo de cómo centralizar datos de diversas fuentes, como CRM y POS, y convertirlos en información valiosa para la toma de decisiones estratégicas, así como en la creación de visualizaciones que faciliten el acceso rápido a insights clave.

3.3.3 Tableau

Tableau es una plataforma de visualización de datos que destaca por su capacidad para crear dashboards interactivos que permiten a los usuarios explorar grandes volúmenes de información de manera dinámica. Utilizada en sectores como finanzas y retail, facilita la representación gráfica de tendencias y patrones a través de gráficos avanzados (Tableau, s.f.).

La forma en que Tableau representa gráficamente sus datos es un gran modelo para seguir en el proyecto, ya que permite a los usuarios navegar y analizar la información de manera ágil y precisa. Esto es fundamental para optimizar la toma de decisiones basada en datos.

4 Metodología

El desarrollo del software se fundamenta en una combinación estratégica de metodologías ágiles, seleccionadas específicamente para adaptarse a las necesidades del proyecto. En lugar de seguir un único marco de trabajo, se integraron principios y prácticas de Extreme Programming (XP) y Scrumban, con el objetivo de optimizar el proceso de desarrollo, priorizando la calidad del software, la adaptabilidad y la entrega incremental de valor.

4.1 Scrumban

Scrumban es una metodología híbrida que combina elementos de Scrum y Kanban, integrando las ceremonias estructuradas de Scrum con los aspectos visuales, los límites de trabajo en progreso (WIP), los sistemas pull y el flujo continuo característicos de Kanban. Esta combinación resulta especialmente útil para equipos o empresas que no han adoptado previamente metodologías ágiles, pero que desean migrar hacia un modelo de flujo de trabajo basado en el enfoque pull (Kanban Tool, 2024).

A diferencia de metodologías más rígidas, Scrumban permite al equipo adaptar su enfoque según las necesidades específicas del proyecto, eliminando o ajustando prácticas que no se consideren esenciales. Esto implica que, dependiendo del contexto, es posible prescindir de ciertas etapas, como la reunión diaria (Daily Scrum), o incluso de roles definidos, como el Scrum Master, manteniendo la flexibilidad sin comprometer la eficiencia del desarrollo.

4.1.1 Conceptos de Scrum

Los conceptos fundamentales de la metodología Scrum (roles, eventos y artefactos) que se implementaron en el proyecto son:

4.1.1.1 Product Owner

El Product Owner tiene la responsabilidad de procurar que el equipo tenga una comprensión de la visión y el propósito del producto con el objetivo de generar valor para todas las partes interesadas, tanto internas como externas a la organización, así como para los usuarios (Schwaber y Sutherland, 2020).

4.1.1.2 Sprints

Los Sprints son el núcleo de Scrum, donde las ideas se transforman en valor. Se trata de eventos de duración fija —de un mes como máximo— que buscan establecer consistencia en el proceso de desarrollo. Un nuevo Sprint comienza inmediatamente después de la finalización del anterior. Todas las actividades necesarias para alcanzar el Objetivo del Producto, como la planificación del Sprint (Sprint Planning) y la revisión del Sprint (Sprint Review), se llevan a cabo dentro de cada Sprint (Schwaber y Sutherland, 2020).

4.1.1.3 Sprint Planning

La planificación del Sprint (Sprint Planning) da inicio al Sprint al definir el trabajo que se realizará durante el mismo. El Scrum Team elabora este plan de forma colaborativa. El Product Owner se encarga de que los participantes estén preparados para discutir los elementos más relevantes del Product Backlog y su relación con el Objetivo del Producto. Asimismo, el Scrum Team puede invitar a otras personas a participar en la planificación con el fin de aportar asesoramiento cuando sea necesario (Schwaber y Sutherland, 2020).

4.1.1.4 Sprint Review

El propósito de la revisión del Sprint (Sprint Review) es inspeccionar los resultados obtenidos durante el Sprint y definir posibles adaptaciones futuras. El Scrum Team presenta el trabajo realizado a los interesados clave, con quienes se analiza el progreso alcanzado respecto al Objetivo del Producto. Durante este evento, el equipo y los interesados revisan tanto lo logrado como los cambios ocurridos en el entorno del proyecto. Con base en esta información, se colabora para determinar los próximos pasos a seguir. El Product Backlog también puede ajustarse para aprovechar nuevas oportunidades. La Sprint Review es una sesión de trabajo activa, y el Scrum Team debe evitar reducirla a una simple presentación (Schwaber y Sutherland, 2020).

4.1.1.5 Sprint Retrospective

El propósito de la retrospectiva del Sprint (Sprint Retrospective) es planificar acciones orientadas a mejorar la calidad y la efectividad del equipo. El Scrum Team inspecciona cómo se desarrolló el Sprint anterior en relación con las personas, las interacciones, los procesos, las herramientas y su Definición de Terminado (Definition of Done). Los elementos evaluados pueden variar según el tipo de trabajo realizado. Se identifican los supuestos que condujeron a errores o desviaciones y se analizan sus causas. Asimismo, el equipo reflexiona sobre los aspectos que funcionaron correctamente, los problemas que surgieron y cómo fueron abordados o, en su defecto, no resueltos (Schwaber y Sutherland, 2020).

4.1.1.6 Product Backlog

El Product Backlog es una lista dinámica y priorizada de los elementos necesarios para mejorar el producto. Representa la única fuente del trabajo que será realizado por el Scrum Team.

Los elementos del Product Backlog que el equipo considera factibles de completar dentro de un Sprint se consideran listos para ser seleccionados durante el evento de Sprint Planning. Generalmente, alcanzan este nivel de claridad tras el proceso de refinamiento. El refinamiento del Product Backlog consiste en descomponer y definir con mayor detalle los elementos del backlog, dividiéndolos en unidades más pequeñas y precisas. Se trata de una actividad continua en la que se agregan detalles como descripciones, prioridades y estimaciones de tamaño. Los atributos específicos de los elementos pueden variar según el contexto y el tipo de trabajo (Schwaber y Sutherland, 2020).

4.1.1.7 Sprint Backlog

El Sprint Backlog es un plan elaborado por y para los desarrolladores. Representa una visión precisa, actualizada en tiempo real, del trabajo que el equipo de desarrollo planea ejecutar durante el Sprint con el fin de alcanzar el Objetivo del Sprint. Por su naturaleza dinámica, el Sprint Backlog se actualiza continuamente a lo largo del Sprint, en función del aprendizaje y de los descubrimientos realizados durante su ejecución (Schwaber y Sutherland, 2020).

4.1.2 Conceptos de Kanban

Los conceptos de la metodología Kanban (roles, eventos y artefactos) que se implementaron en el proyecto son:

4.1.2.1 Tablero Kanban

Un Tablero Kanban es una herramienta visual, ya sea física (como una pizarra de pared) o digital, que el equipo utiliza para representar el flujo de trabajo. Este se organiza en columnas que

reflejan las distintas etapas del proceso, y puede incluir filas que indican los diferentes proyectos o clientes que se están gestionando simultáneamente (Kanban Tool, 2024).

4.1.2.2 Tickets Kanban

Las tarjetas Kanban representan las unidades individuales de trabajo que fluyen a través del software. Cada tarjeta puede contener información relevante sobre la tarea, como asignaciones, prioridad, estado, entre otros datos (Kanban Tool, 2024).

4.1.2.3 Work in progress

El límite de trabajo en curso (WIP, por sus siglas en inglés) se establece para cada columna del tablero Kanban. Este límite contribuye a controlar y gestionar la cantidad de tareas que se realizan simultáneamente, evitando la sobrecarga del equipo (Kanban Tool, 2024).

4.2 Extreme Programming (XP)

Extreme Programming (XP) es uno de los marcos ágiles más reconocidos y aplicados en la industria del desarrollo de software. Se caracteriza por su fuerte enfoque en los aspectos técnicos del desarrollo, priorizando la escritura de código de alta calidad en plazos cortos. Su objetivo principal es encontrar formas eficientes de desarrollar software, manteniéndose adaptable ante los cambios constantes en los requisitos del cliente (AltexSoft Editorial Team, 2021).

Para alcanzar este objetivo, Extreme Programming (XP) propone un conjunto de prácticas de ingeniería que permiten a los desarrolladores mejorar su rendimiento y eficacia, potenciando sus capacidades técnicas. En el marco de este proyecto, se han seleccionado las siguientes prácticas para su aplicación:

4.2.1 Refactorización

Consiste en mejorar la estructura, legibilidad, eficiencia y seguridad del programa sin alterar su funcionalidad. Cada desarrollador debe refactorizar su código en cuanto identifique una posible mejora, con el objetivo de mantener una base de código limpia, escalable y fácil de mantener (AltexSoft Editorial Team, 2021).

4.2.2 Estándares de código

El equipo debe establecer y aplicar prácticas comunes de codificación, utilizando formatos y estilos uniformes al escribir el código. La adopción de estándares facilita que los miembros del equipo puedan leer, compartir y refactorizar el código con mayor facilidad, además de permitir el seguimiento de contribuciones individuales y acelerar el proceso de aprendizaje para nuevos integrantes. Un código escrito bajo un conjunto de reglas coherentes promueve la propiedad colectiva del proyecto (AltexSoft Editorial Team, 2021).

4.2.3 Diseño simple

El mejor diseño para un software es aquel que sea lo más simple posible y que funcione correctamente. Si se detecta alguna complejidad innecesaria, debe ser eliminada. Un diseño adecuado debe superar las pruebas implementadas, evitar la duplicación de código y mantener el menor número posible de métodos y clases. Además, debe expresar de forma precisa la intención del programador (AltexSoft Editorial Team, 2021).

4.2.4 Metáfora del sistema

La metáfora del sistema representa un diseño sencillo que reúne ciertas cualidades clave. En primer lugar, la estructura del software debe ser comprensible para cualquier persona nueva en el equipo, permitiéndole integrarse y comenzar a trabajar sin necesidad de revisar a fondo toda la documentación. En segundo lugar, los nombres de las clases y métodos deben ser coherentes y significativos. Los desarrolladores deben procurar nombrar los elementos del software como si ya existieran en el lenguaje cotidiano, lo que facilita la comprensión general del diseño (AltexSoft Editorial Team, 2021).

5 Desarrollo del Proyecto

Se presenta el diseño del software, el cual incluye los requisitos funcionales y no funcionales, los casos de uso, los diagramas de actividades, el diseño de la interfaz gráfica y el modelo de base de datos.

5.1 Análisis y Requerimientos

A continuación, se enumeran y describen los requerimientos funcionales y no funcionales de la aplicación, los cuales fueron definidos tanto al inicio como durante el desarrollo del proyecto.

Los requerimientos finales se establecieron a través de reuniones y discusiones con el tutor del proyecto.

5.1.1 Requerimientos Funcionales

En esta sección se describen las funciones que debe cumplir el software.

Tabla 1

Requerimiento funcional: Iniciar sesión

Requerimiento Funcional	
Requerimiento	RF01
Nombre	Iniciar sesión
Prioridad	Alta
Descripción	Los usuarios registrados podrán iniciar sesión en el sitio usando el nit de la empresa asignada, su cedula y contraseña.

Tabla 2

Requerimiento funcional: Cerrar sesión

Requerimiento Funcional	
Requerimiento	RF02
Nombre	Cerrar sesión
Prioridad	Alta
Descripción	Permitir a los usuarios cerrar sesión, eliminando su token de autenticación del dispositivo al que estaba conectado.

Tabla 3

Requerimiento funcional: Edición de cuenta

Requerimiento Funcional	
Requerimiento	RF03
Nombre	Edición de cuenta
Prioridad	Alta
Descripción	Permitir a los usuarios registrados poder cambiar los datos con los cuales se registraron, siendo nombre, correo electrónico, cedula y su rol, si posee permiso de cambiarlo.

Tabla 4

Requerimiento funcional: Cambio de contraseña

Requerimiento Funcional	
Requerimiento	RF04
Nombre	Cambio de contraseña
Prioridad	Alta

Descripción	Permitir a los usuarios registrados poder cambiar su contraseña, deberán saber su contraseña anterior para realizar dicha acción.

Tabla 5

Requerimiento funcional: Recuperación de contraseña

Requerimiento Funcional	
Requerimiento	RF05
Nombre	Recuperación de contraseña
Prioridad	Alta
Descripción	Permitir a los usuarios registrados el poder solicitar el cambio de su contraseña en caso de haberla olvidado, utilizando su correo electrónico para recibir un mensaje que los redirigirá a un formulario donde podrán crear una nueva contraseña.

Tabla 6

Requerimiento funcional: Autorización

Requerimiento Funcional	
Requerimiento	RF06
Nombre	Autorización
Prioridad	Alta
Descripción	El software debe verificar que los usuarios cuenten con la autorización correspondiente representada con los permisos que posee su rol al intentar acceder a cada funcionalidad específica.

Tabla 7

Requerimiento funcional: Autenticación

Requerimiento Funcional	
Requerimiento	RF07
Nombre	Autenticación
Prioridad	Alta
Descripción	El software debe verificar que los usuarios estén autenticados antes de permitirles acceso a las funcionalidades.

Tabla 8

Requerimiento funcional: Gestión de roles

Requerimiento Funcional	
Requerimiento	RF08
Nombre	Gestión de roles
Prioridad	Alta
Descripción	El software debe permitir que los usuarios autorizados gestionen roles, incluyendo su creación, edición y eliminación.

Tabla 9

Requerimiento funcional: Gestión de usuarios

Requerimiento Funcional	
Requerimiento	RF09
Nombre	Gestión de usuarios
Prioridad	Alta

Descripción	El software debe permitir que los usuarios autorizados gestiones usuarios, incluyendo su creación, edición y eliminación.

Tabla 10

Requerimiento funcional: Confirmación de cuenta

Requerimiento Funcional	
Requerimiento	RF10
Nombre	Confirmación de cuenta
Prioridad	Alta
Descripción	El software debe de enviar un mensaje al correo electrónico del usuario creado para confirmar su cuenta, de lo contrario no podrá ingresar a la plataforma.

Tabla 11

Requerimiento funcional: Vista de bienvenida

Requerimiento Funcional	
Requerimiento	RF11
Nombre	Vista de bienvenida
Prioridad	Media
Descripción	El software debe de llevar a los usuarios a una vista de bienvenida una vez inicien sesión correctamente, dando una breve introducción acerca de la plataforma, esta vista debe poder ser accedida por todo tipo de usuario.

Tabla 12

Requerimiento funcional: Inclusión de proceso ETL

Requerimiento Funcional

Requerimiento	RF12
Nombre	Inclusión de proceso ETL
Prioridad	Alta
Descripción	El software debe incluir un proceso ETL (Extracción, Transformación y Carga) que permita obtener datos desde fuentes externas, transformarlos según las necesidades del software y cargarlos en la base de datos para su posterior análisis o uso en reportes e indicadores clave.

Tabla 13

Requerimiento funcional: Cálculo y visualización de indicadores (KPI)

Requerimiento Funcional	
Requerimiento	RF13
Nombre	Cálculo y visualización de indicadores (KPI)
Prioridad	Alta
Descripción	El software debe calcular y mostrar 16 indicadores clave de rendimiento (KPI): Total de Ventas, Precio Total de Ventas, Precio Útil Total de Ventas, Margen de Ganancia, Valor del Ticket Promedio, Evolución de Ventas Totales Mensuales, Evolución de los Ingresos Totales de Utilidad en Ventas, Evolución de Ingresos Totales de Ventas, Evolución de Margen de Ganancias Totales, Evolución del Valor de Ticket Promedio Totales, Ventas Totales por Sucursal, Top X Artículos Más Vendidos, Top X Artículos Menos Vendidos, Ventas Totales del Último Año, Comparativa Trimestral de Ventas, y Crecimiento Anual de Ventas Totales.

Tabla 14

Requerimiento funcional: Prototipo de predicción de ventas

Requerimiento Funcional	
Requerimiento	RF14
Nombre	Prototipo de predicción de ventas

Prioridad	Alta
Descripción	Se debe presentar una propuesta de prototipo para un modelo de predicción de ventas diarias de productos. Este modelo no estará integrado en el software final, sino que se desarrollará de forma independiente en un entorno aparte.

Tabla 15

Requerimiento funcional: Persistencia de sesión iniciada

Requerimiento Funcional	
Requerimiento	RF15
Nombre	Persistencia de sesión iniciada
Prioridad	Alta
Descripción	El software debe de mantener la sesión iniciada incluso después de cerrar el navegador o recargar la página. Esta funcionalidad debe implementarse de manera segura, mediante tokens persistentes con expiración y mecanismos de protección contra accesos no autorizados.

Tabla 16

Requerimiento funcional: Modo claro y oscuro

Requerimiento Funcional	
Requerimiento	RF16
Nombre	Modo claro y oscuro
Prioridad	Media
Descripción	El software debe permitir a los usuarios cambiar entre el modo claro y el modo oscuro de la interfaz, con el fin de mejorar la experiencia de uso según sus preferencias visuales y condiciones de iluminación. Esta opción debe estar disponible en la configuración del perfil o en un menú de accesibilidad visible.

5.1.2 Requerimientos no funcionales

En esta sección se describen las características de calidad que debe cumplir el software con el fin de lograr un buen desempeño, usabilidad, mantenibilidad, seguridad y otros aspectos no relacionados directamente con las funcionalidades específicas, pero que son fundamentales para el correcto funcionamiento del software.

Tabla 17

Requerimiento no funcional: Rendimiento y tiempo de respuesta

Requerimiento No Funcional	
Requerimiento	RNF01
Nombre	Rendimiento y tiempo de respuesta
Prioridad	Alta
Descripción	El software debe responder a las solicitudes de los usuarios en un tiempo óptimo, manteniendo altos niveles de rendimiento.

Tabla 18

Requerimiento no funcional: Seguridad y privacidad de los datos

Requerimiento No Funcional	
Requerimiento	RNF02
Nombre	Seguridad y privacidad de los datos
Prioridad	Alta
Descripción	El software debe preservar la seguridad y privacidad de los datos de los usuarios mediante protocolos de cifrado, autenticación robusta y una base de datos segura.

Tabla 19

Requerimiento no funcional: Escalabilidad

Requerimiento No Funcional	
Requerimiento	RNF03
Nombre	Escalabilidad
Prioridad	Alta
Descripción	El software debe ser escalable, permitiendo la adición o modificación de funcionalidades sin afectar el rendimiento o la estabilidad de este.

Tabla 20

Requerimiento no funcional: Integridad y consistencia de los datos

Requerimiento No Funcional	
Requerimiento	RNF04
Nombre	Integridad y consistencia de los datos
Prioridad	Alta
Descripción	El software debe preservar la integridad y consistencia de los datos almacenados y procesados, evitando su pérdida, corrupción o alteración.

Tabla 21

Requerimiento no funcional: Alta disponibilidad

Requerimiento No Funcional	
Requerimiento	RNF05
Nombre	Alta disponibilidad
Prioridad	Alta

Descripción	El software debe mantener un alto nivel de disponibilidad, asegurando que esté operativo la mayor parte del tiempo, salvo en ventanas de mantenimiento programado.

Tabla 22

Requerimiento no funcional: Soporte de acceso concurrente

Requerimiento No Funcional	
Requerimiento	RNF06
Nombre	Soporte de acceso concurrente
Prioridad	Alta
Descripción	El software debe soportar el acceso simultáneo de múltiples usuarios sin comprometer el rendimiento ni la seguridad de la información.

Tabla 23

Requerimiento no funcional: Compatibilidad entre navegadores

Requerimiento No Funcional	
Requerimiento	RNF07
Nombre	Compatibilidad entre navegadores
Prioridad	Media
Descripción	El software debe ser compatible con los principales navegadores web (Chrome, Firefox, Safari y Edge) para asegurar su accesibilidad en diferentes plataformas.

Tabla 24

Requerimiento no funcional: Usabilidad e interfaz intuitiva

Requerimiento No Funcional

Requerimiento	RNF08
Nombre	Usabilidad e interfaz intuitiva
Prioridad	Alta
Descripción	El software debe contar con una interfaz gráfica intuitiva y fácil de usar, diseñada según principios de usabilidad centrada en el usuario.

5.2 Diagramas UML

En esta sección se describe el diseño general de la plataforma, incluyendo los principales elementos que lo componen. Se presentan los diagramas UML utilizados durante el análisis y desarrollo, la arquitectura del software adoptada, el modelo de base de datos implementado y la estructura de los modelos predictivos. Cada uno de estos elementos proporciona una visión detallada del funcionamiento interno del software y cómo sus distintos componentes interactúan entre sí.

Los diagramas UML permiten representar de forma visual distintos aspectos de la plataforma. A continuación, se presentan los diagramas de casos de uso, que describen las funcionalidades principales desde la perspectiva del usuario, y los diagramas de actividades, que detallan el flujo de procesos dentro del software.

5.2.1 Diagrama de Casos de Uso

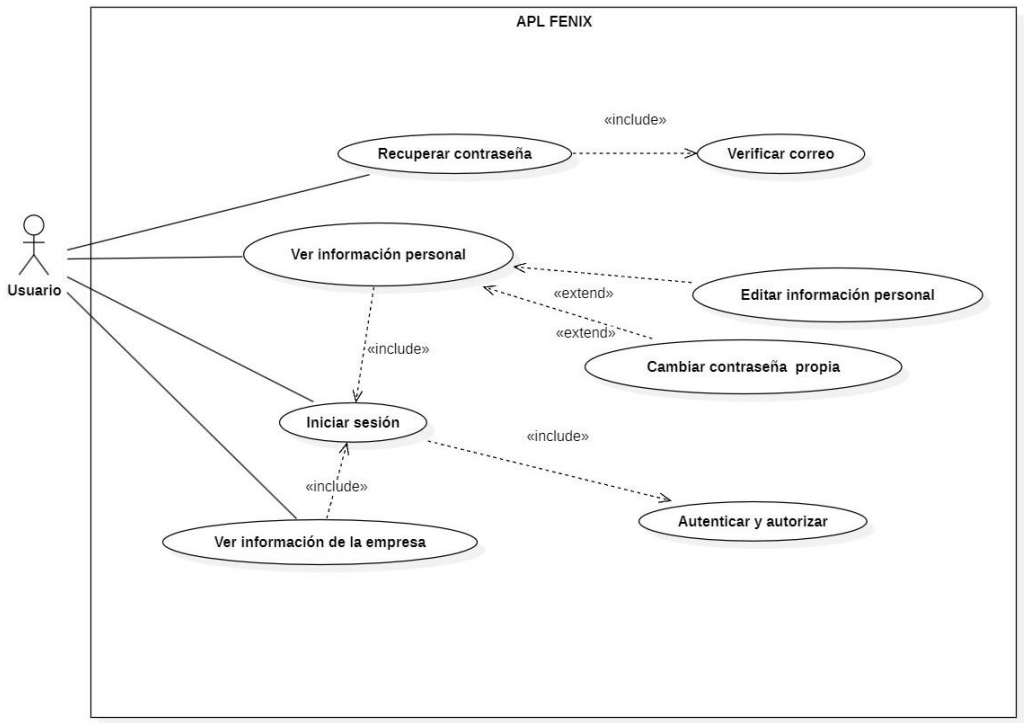
El diagrama de casos de uso es una herramienta fundamental del modelado UML que permite representar las interacciones entre los distintos actores del software y las funcionalidades que este ofrece. A través de este diagrama, se identifican los principales casos de uso que describen

cómo los usuarios finales pueden interactuar con la aplicación, sirviendo como base para el diseño funcional y la validación de requisitos.

La Figura 3 muestra el diagrama de casos de uso correspondiente a las funcionalidades generales de la plataforma, accesibles sin necesidad de autenticación. Entre estas se incluyen acciones como iniciar sesión, recuperar contraseña y visualizar información básica del usuario.

Figura 3

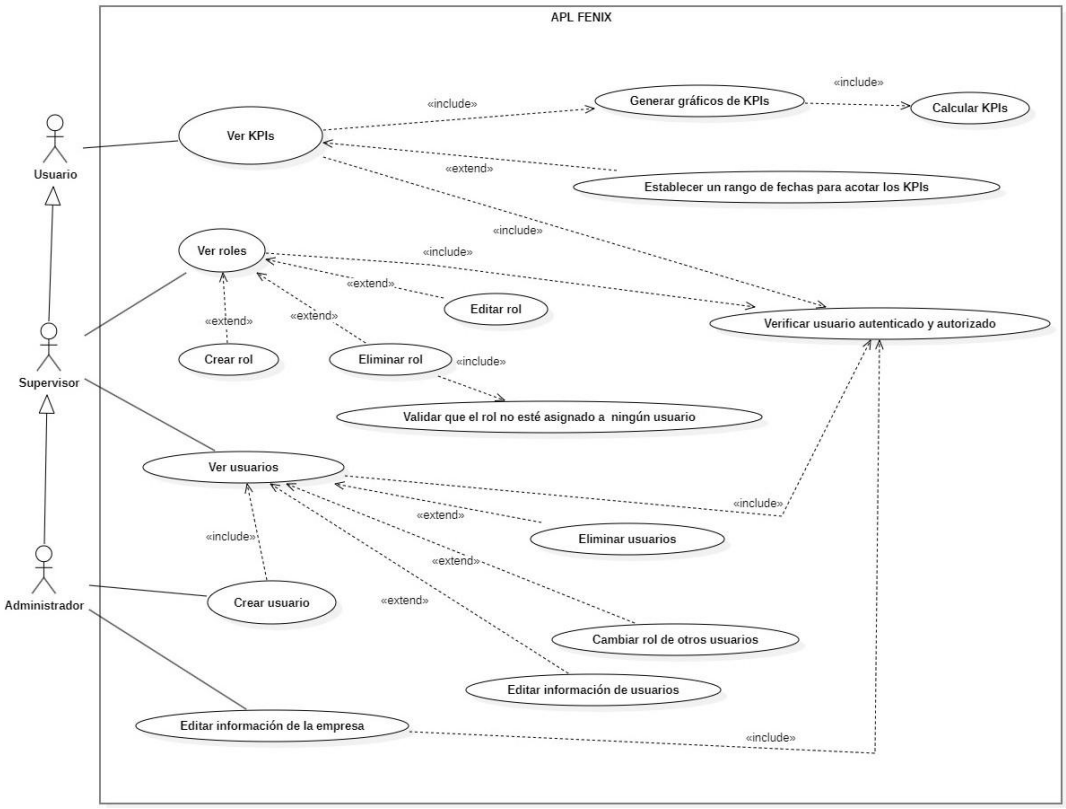
Diagrama de casos de uso - Funcionalidades generales del software



La Figura 4 presenta el diagrama de casos de uso que representa las funcionalidades disponibles para los usuarios autenticados. Este conjunto de casos de uso contempla operaciones como la consulta de KPIs y la gestión de datos maestros.

Figura 4

Diagrama de casos de uso – Funcionalidades para usuarios autenticados



5.2.2 Diagramas de Actividades

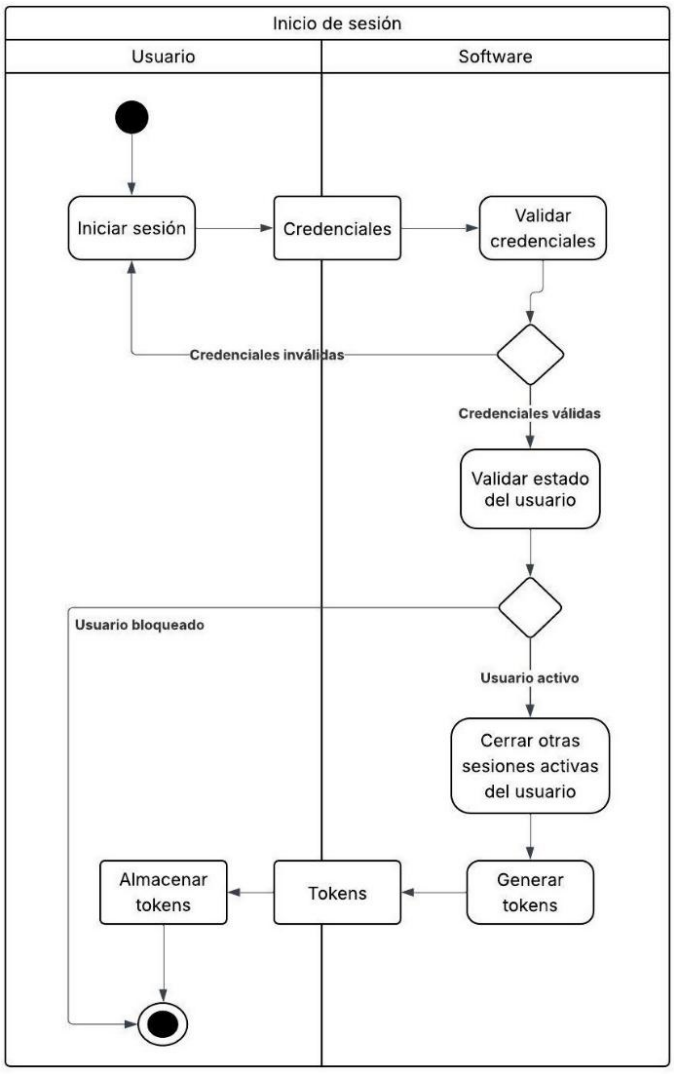
A continuación, se presentan los diagramas de actividades que ilustran el comportamiento de los casos de uso dentro del software, como el inicio de sesión y la consulta de indicadores.

La Figura 5 presenta el diagrama de actividades del proceso de inicio de sesión. El flujo se organiza en dos carriles: uno correspondiente al usuario y otro a la plataforma. El proceso comienza con el ingreso de credenciales por parte del usuario. La plataforma valida la autenticidad de estos datos y, en caso de ser correctos, se procede a verificar si existe una sesión activa previa; de ser así, esta se cierra antes de generar nuevos tokens de acceso y actualización. Finalmente, los

tokens son enviados al usuario para permitir el uso autenticado del software. En caso de que las credenciales no sean válidas, se deniega el acceso.

Figura 5

Diagrama de actividades para inicio de sesión

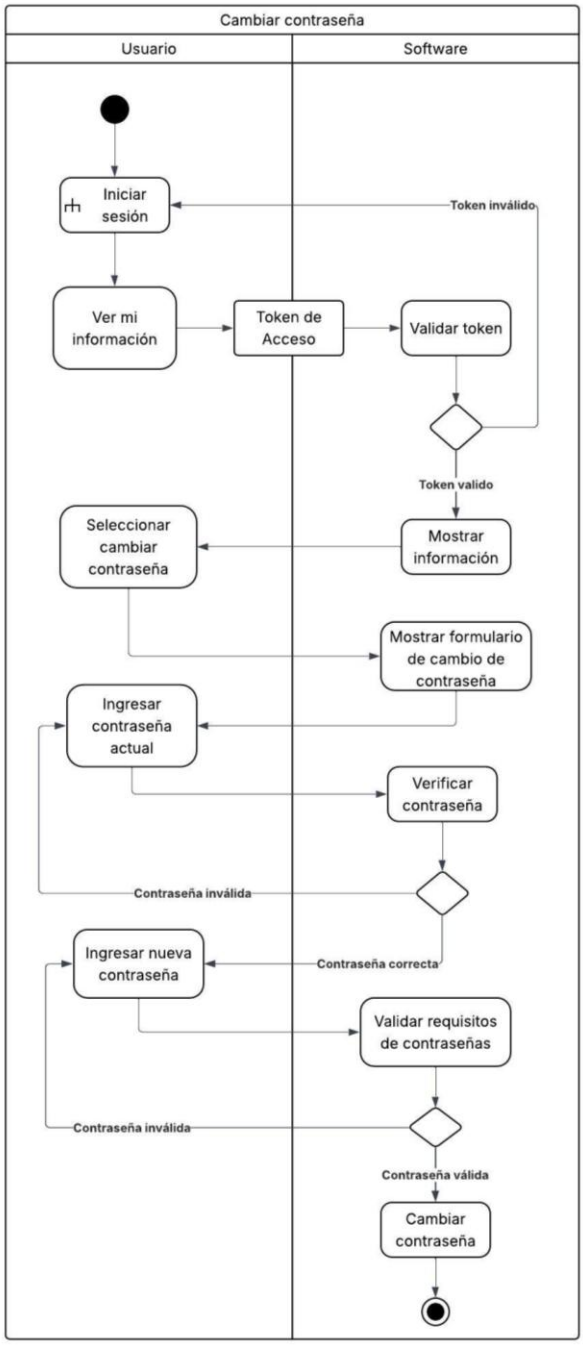


La Figura 6 muestra el diagrama de actividades del proceso de cambio de contraseña. Este proceso requiere que el usuario haya iniciado sesión previamente, utilizando credenciales válidas. Una vez autenticado, el usuario accede a la sección de información de la cuenta, lo que implica el envío y validación de los tokens generados. Posteriormente, se selecciona la opción para cambiar

la contraseña, lo cual activa un formulario donde se solicita la contraseña actual y una nueva. La plataforma verifica la validez de la contraseña actual y, en caso de que la nueva contraseña cumpla con los criterios establecidos, se actualiza la información correspondiente.

Figura 6

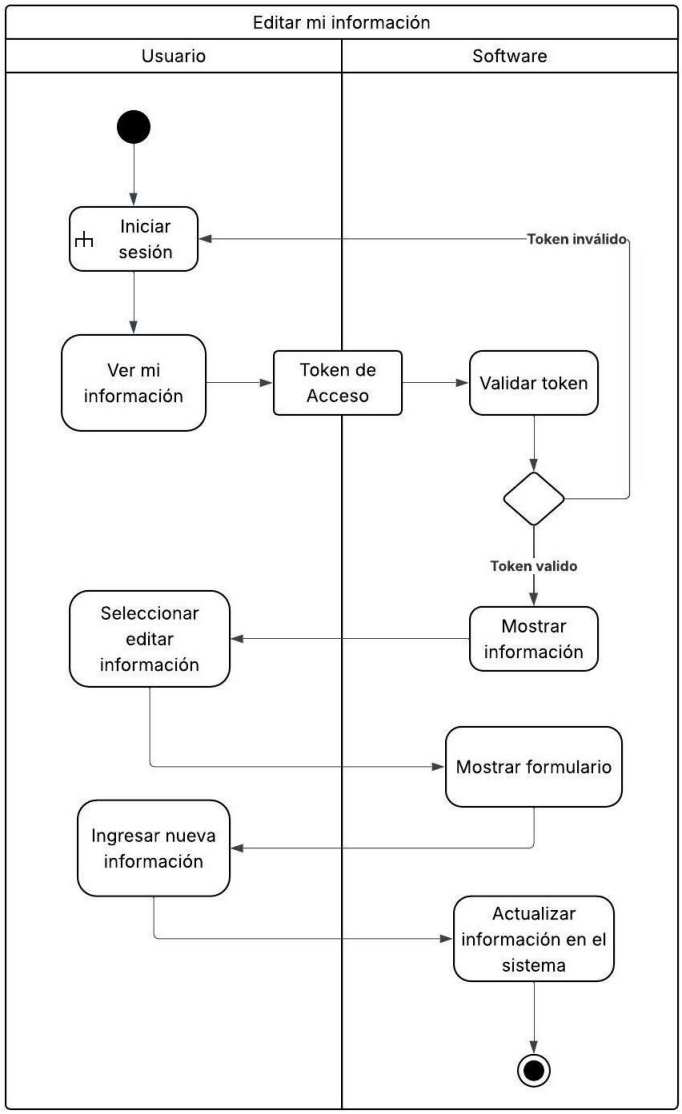
Diagrama de actividades para cambiar contraseña



La Figura 7 representa el diagrama de actividades correspondiente al proceso de edición de información personal. El usuario debe iniciar sesión y acceder a la sección de su información. En este punto, el software valida el token de acceso para verificar la autenticidad de la solicitud. Si el token es válido, la plataforma muestra los datos actuales del usuario y habilita la opción de edición. Una vez seleccionada esta opción, se despliega un formulario para modificar los datos. Finalmente, al ingresar la nueva información, el software la actualiza en la base de datos.

Figura 7

Diagrama de actividades para editar información personal

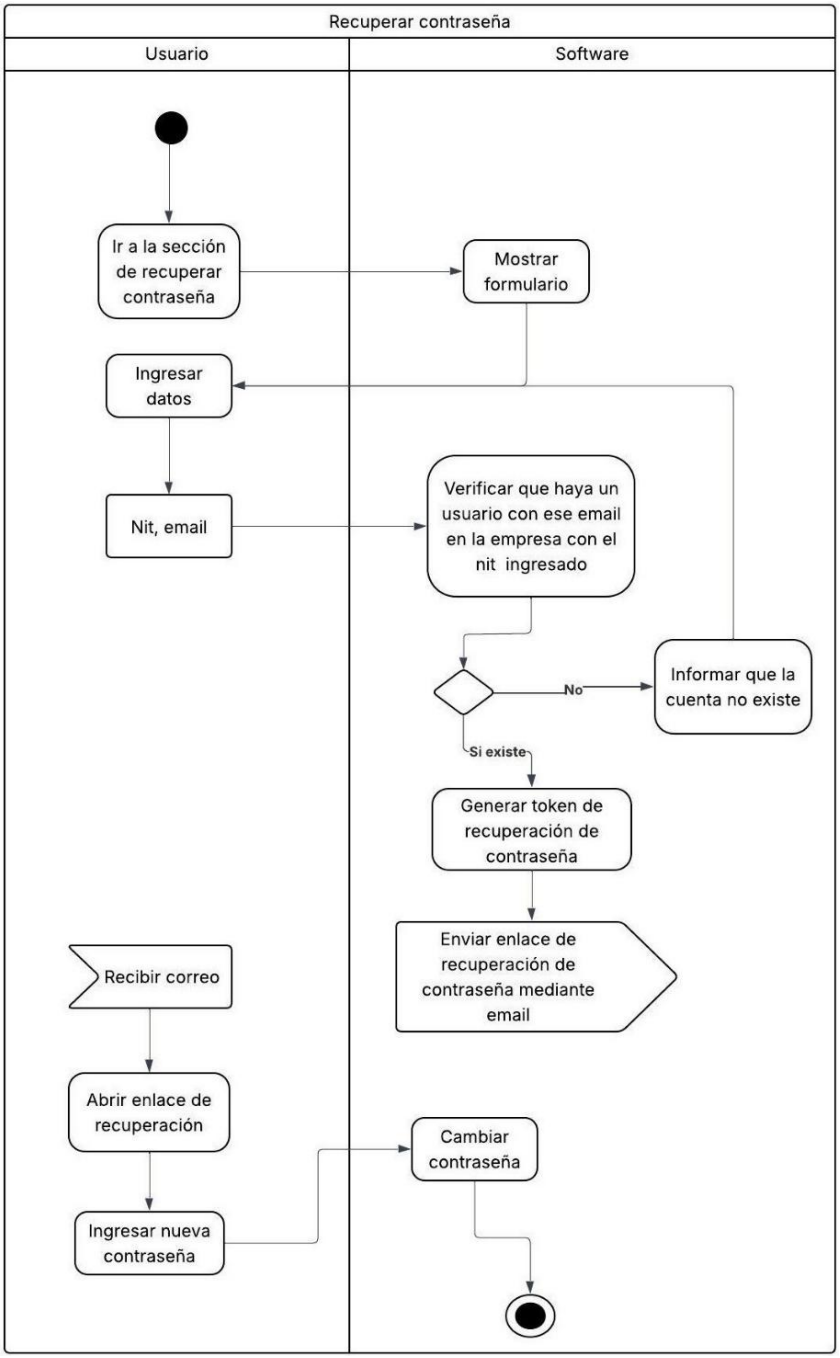


La Figura 8 presenta el diagrama de actividades asociado al proceso de recuperación de contraseña. El usuario inicia el proceso accediendo a la sección correspondiente, donde el software muestra un formulario para ingresar los datos requeridos: NIT de la empresa y correo electrónico. Una vez enviados, la plataforma verifica si existe un usuario registrado con esos datos. Si no se encuentra coincidencia, se notifica que la cuenta no existe. En caso contrario, se genera un token

de recuperación y se envía un enlace por correo electrónico. El usuario accede al enlace, ingresa la nueva contraseña y el software la actualiza en la base de datos.

Figura 8

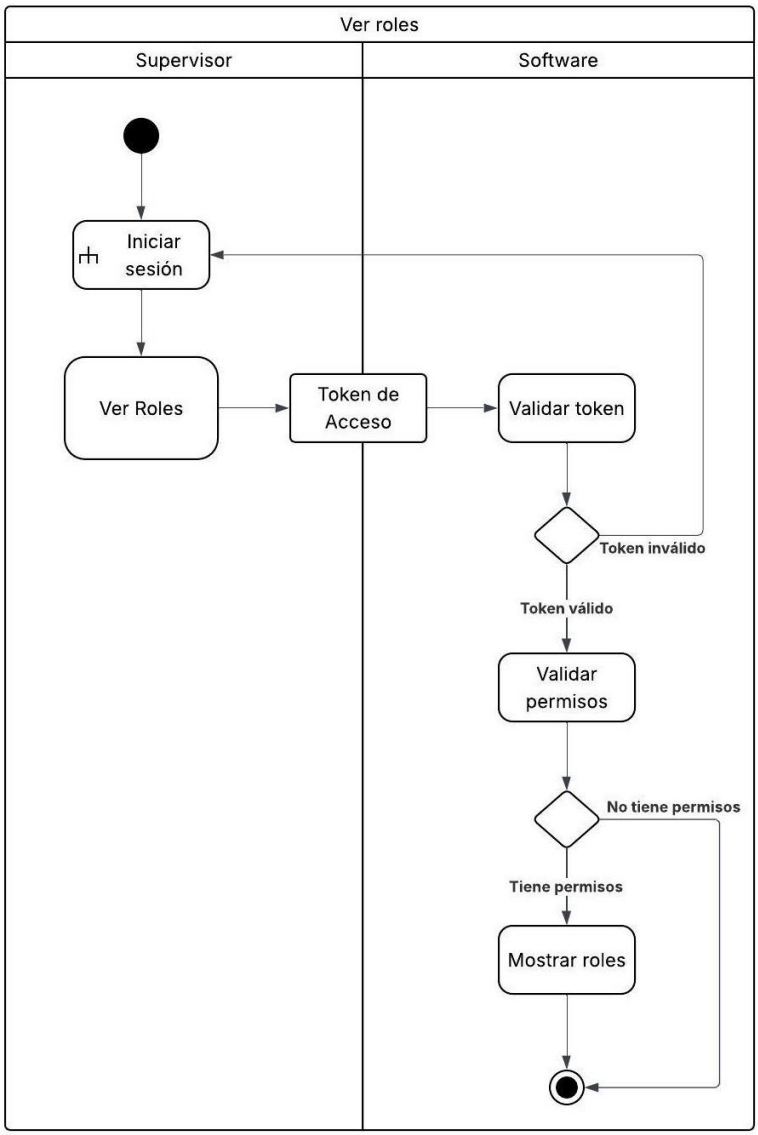
Diagrama de actividades para la recuperación de contraseña



La Figura 9 muestra el diagrama de actividades para el proceso de consulta de roles por parte de un supervisor. Luego de iniciar sesión, el usuario solicita visualizar los roles. El software recibe el token de acceso y procede a validarlo. Si el token es inválido, se detiene el proceso. Si es válido, se verifica si el usuario tiene los permisos necesarios. En caso de no contar con los permisos requeridos, no se muestran resultados. Si los permisos son adecuados, el software presenta la información correspondiente a los roles registrados.

Figura 9

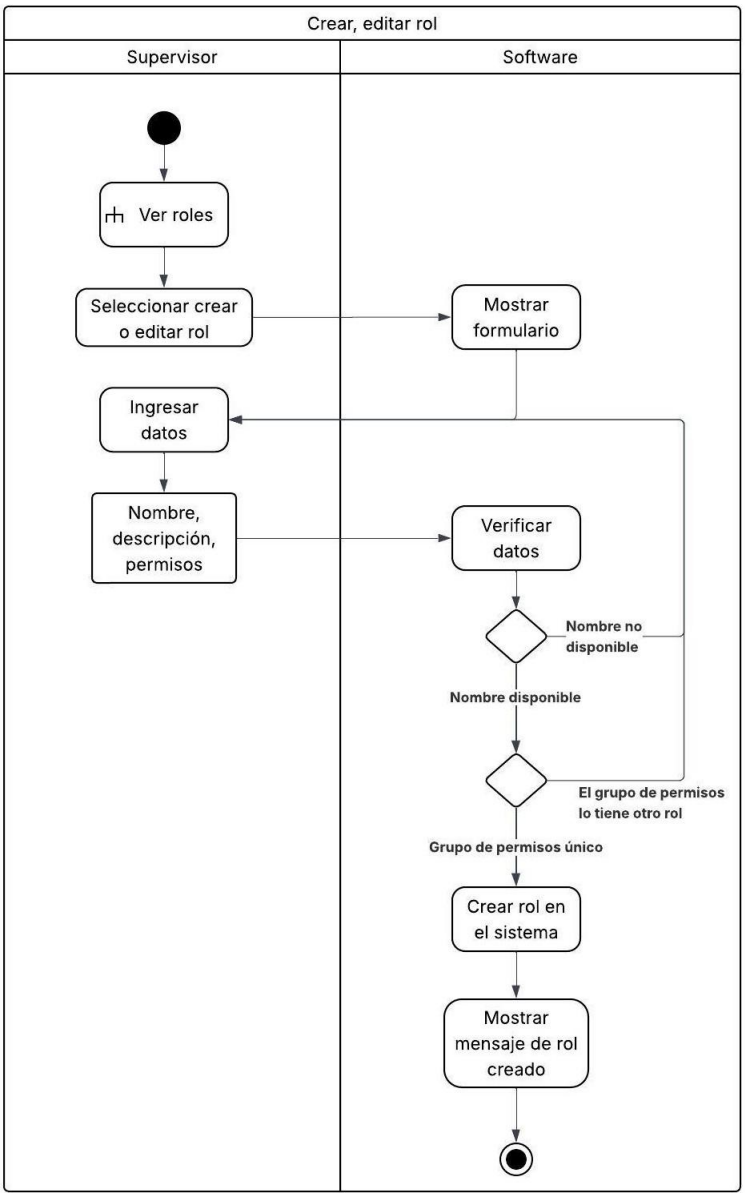
Diagrama de actividades para ver los roles



La Figura 10 representa el flujo de actividades para la creación o edición de un rol por parte de un supervisor. El proceso inicia con la consulta de los roles existentes y la selección de la opción para crear o modificar uno. Luego, el software muestra el formulario correspondiente. El usuario ingresa el nombre, la descripción y el conjunto de permisos. El software verifica que el nombre del rol esté disponible y que el conjunto de permisos no coincida exactamente con el de un rol existente. Si ambas condiciones se cumplen, se procede a registrar el nuevo rol y se muestra un mensaje de confirmación al usuario.

Figura 10

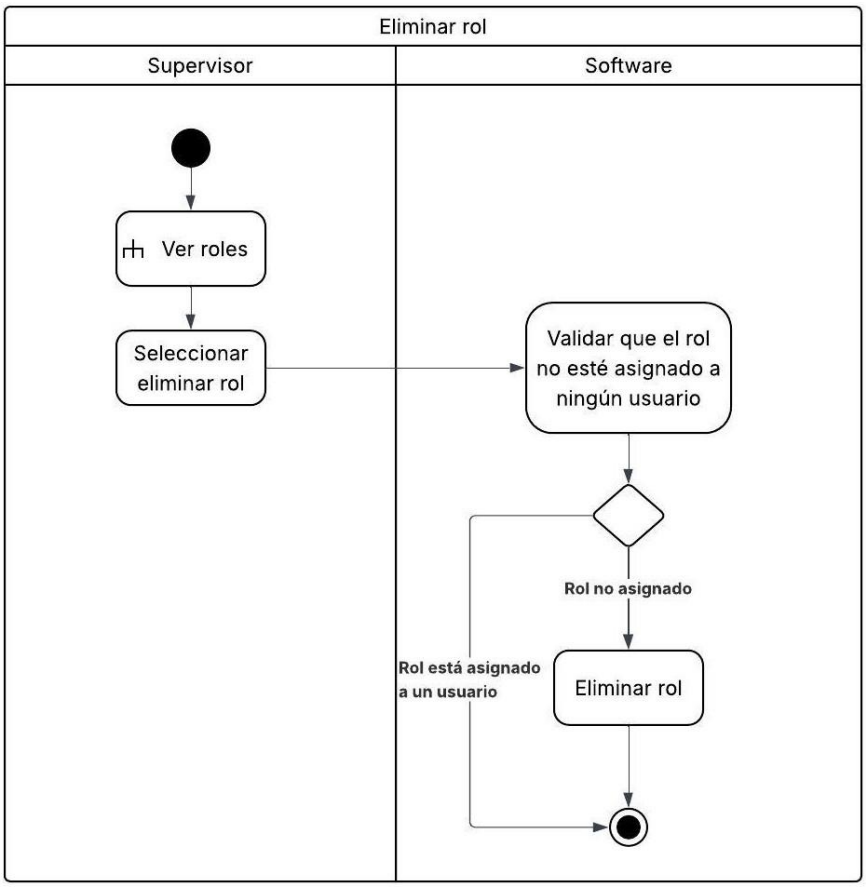
Diagrama de actividades para crear y editar roles



La Figura 11 describe el proceso para eliminar un rol dentro del software. El flujo comienza con la visualización de los roles por parte del supervisor, quien selecciona la opción para eliminar uno de ellos. Posteriormente, el software verifica que el rol no esté asignado a ningún usuario. Si la condición se cumple, el rol es eliminado. En caso contrario, no se realiza ninguna modificación.

Figura 11

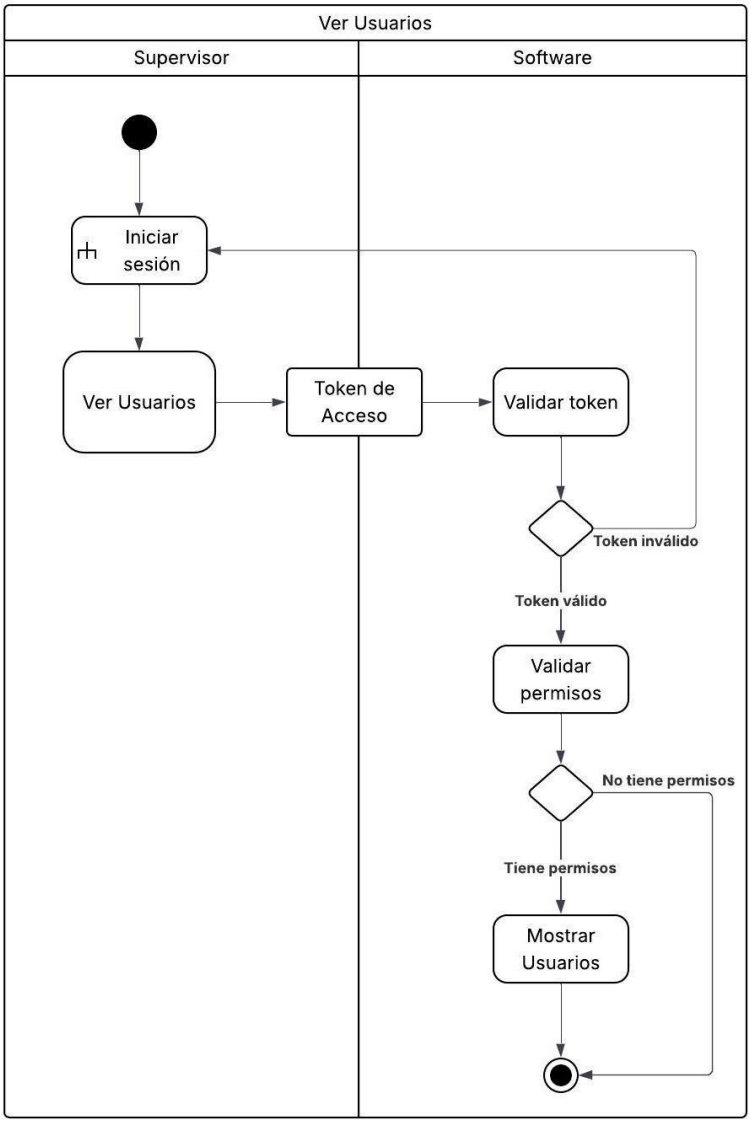
Diagrama de actividades para eliminar un rol



La Figura 12 presenta el flujo de actividades para consultar la lista de usuarios registrados. El proceso inicia con el acceso del supervisor al software. Una vez autenticado, se emite un token de acceso que es validado por el software. Si el token es válido, se procede a verificar si el usuario tiene los permisos requeridos. En caso de que los permisos sean suficientes, se muestra la información correspondiente a los usuarios registrados.

Figura 12

Diagrama de actividades para ver usuarios

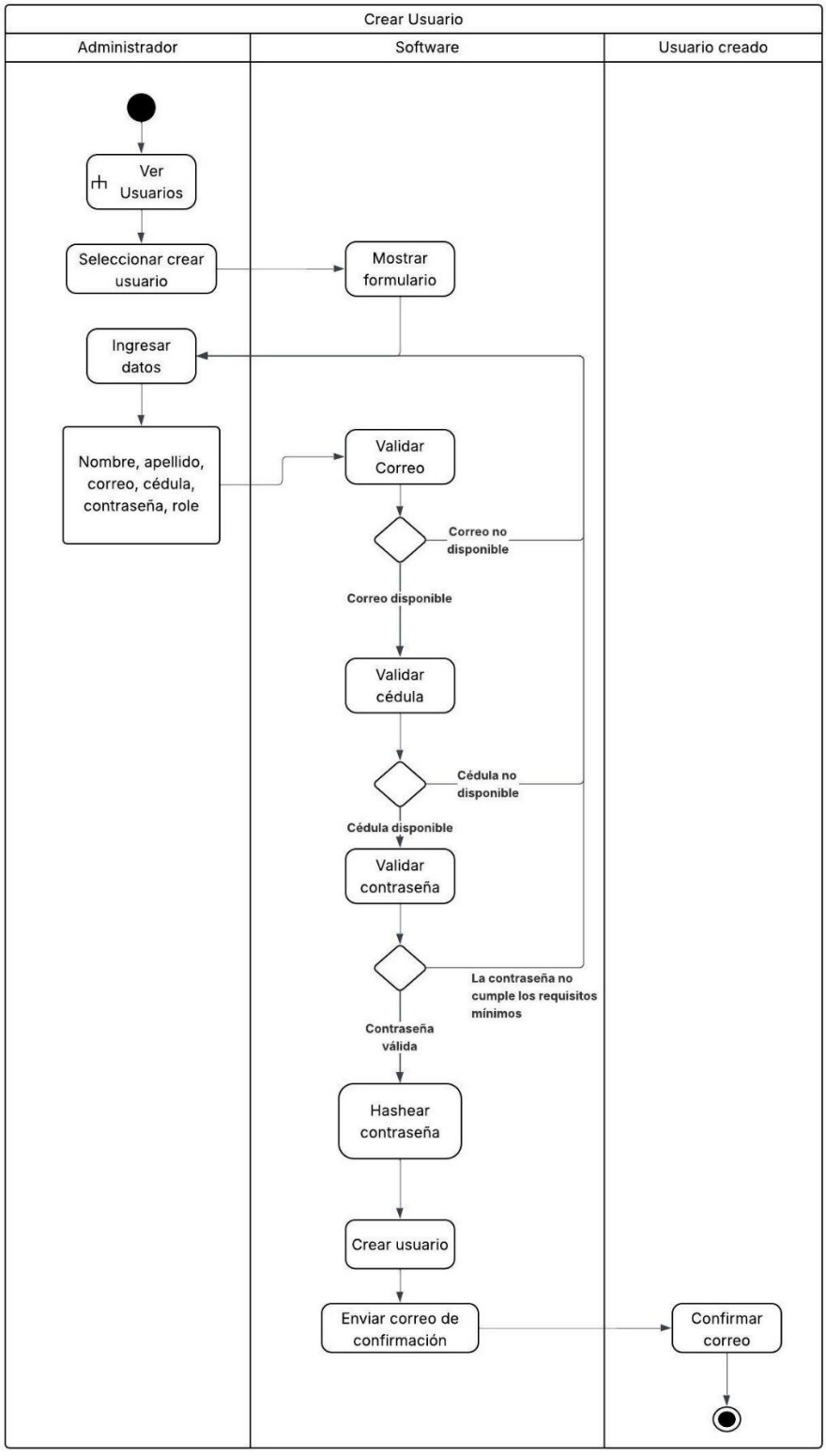


La Figura 13 muestra el proceso que permite al administrador registrar nuevos usuarios en la plataforma. El flujo inicia con la visualización de los usuarios existentes, seguida de la selección de la opción para crear uno nuevo. El software presenta un formulario en el cual se deben ingresar los datos requeridos: nombre, apellidos, correo, número de cédula, contraseña y rol asignado.

Luego de recibir la información, el software ejecuta validaciones secuenciales para verificar que el correo y la cédula no se encuentren registrados, y que la contraseña cumpla los criterios establecidos. Si los datos son válidos, se aplica un proceso de hash a la contraseña y se procede con la creación del usuario. Como último paso, se envía un correo de confirmación para completar el registro, el cual debe ser confirmado por la persona correspondiente.

Figura 13

Diagrama de actividades para crear usuario



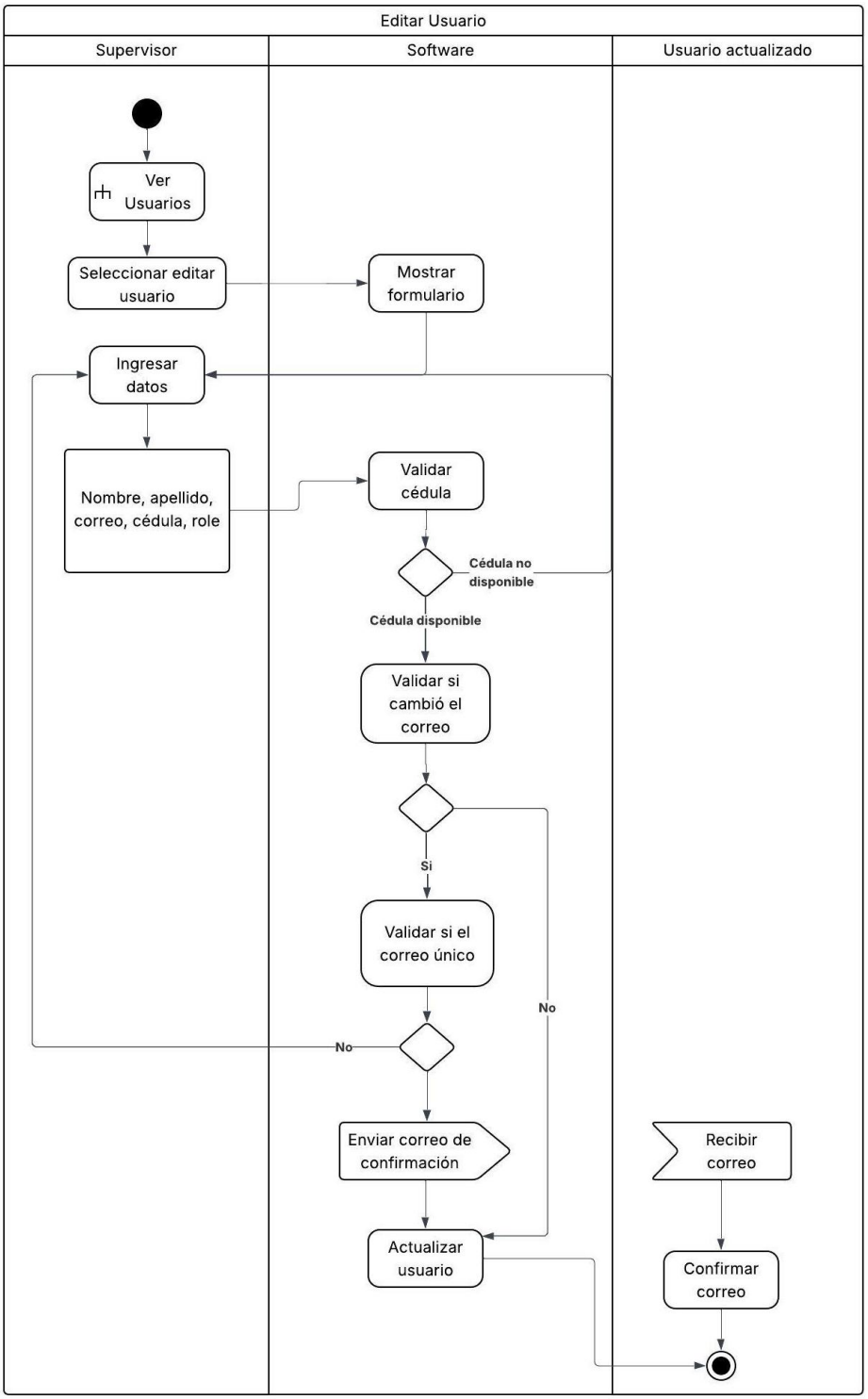
La Figura 14 representa el flujo de actividades involucradas en la modificación de los datos de un usuario existente por parte del supervisor. El proceso se inicia con la visualización de los usuarios, seguida de la selección de la opción para editar uno de ellos. A continuación, el software despliega un formulario con los campos editables.

Una vez se ingresan los nuevos datos, el software realiza una serie de validaciones. Primero, verifica si el número de cédula sigue siendo único dentro del software. Luego, en caso de que el correo haya sido modificado, se evalúa su unicidad para evitar duplicidades. Si ambas condiciones se cumplen, se procede con la actualización de la información.

Como paso final, se envía un correo electrónico al usuario actualizado, quien debe confirmar el cambio de dirección para completar el proceso.

Figura 14

Diagrama de actividades para editar usuario

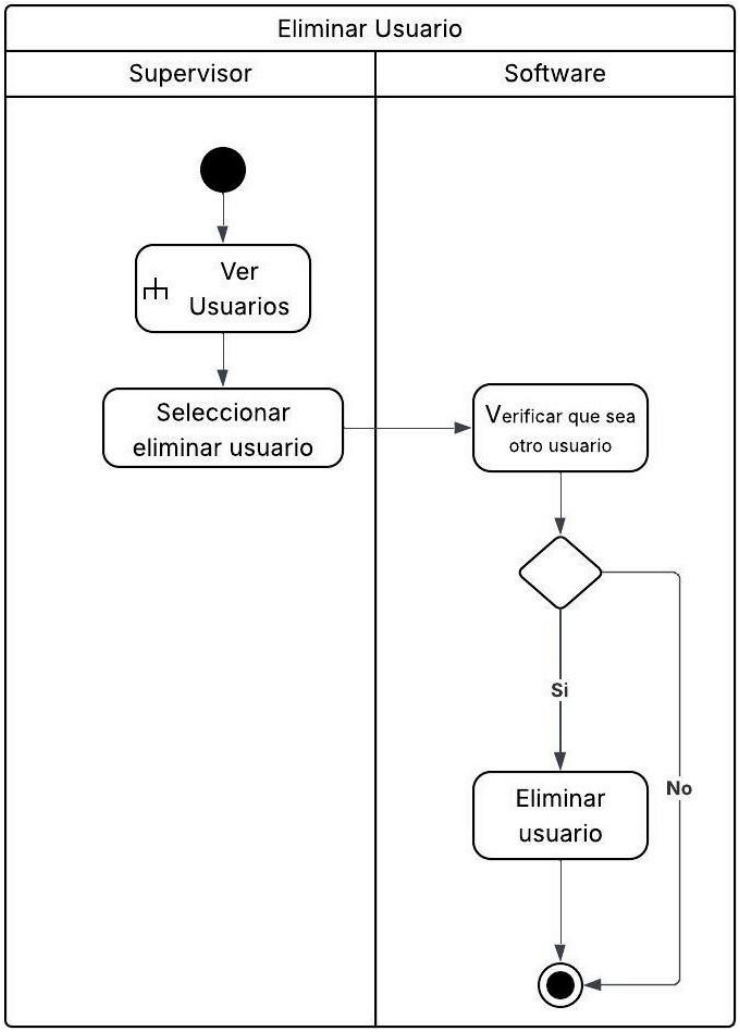


La Figura 15 ilustra el flujo de actividades requerido para eliminar un usuario desde la interfaz del supervisor. El proceso se inicia con la visualización del listado de usuarios, tras lo cual se selecciona la opción para eliminar uno de ellos.

Antes de proceder con la eliminación, el software ejecuta una validación para comprobar que el usuario seleccionado no sea el mismo que está realizando la operación. Si se verifica que es un usuario distinto, permite completar el proceso de eliminación; de lo contrario, la operación no se lleva a cabo.

Figura 15

Diagrama de actividades para eliminar usuario

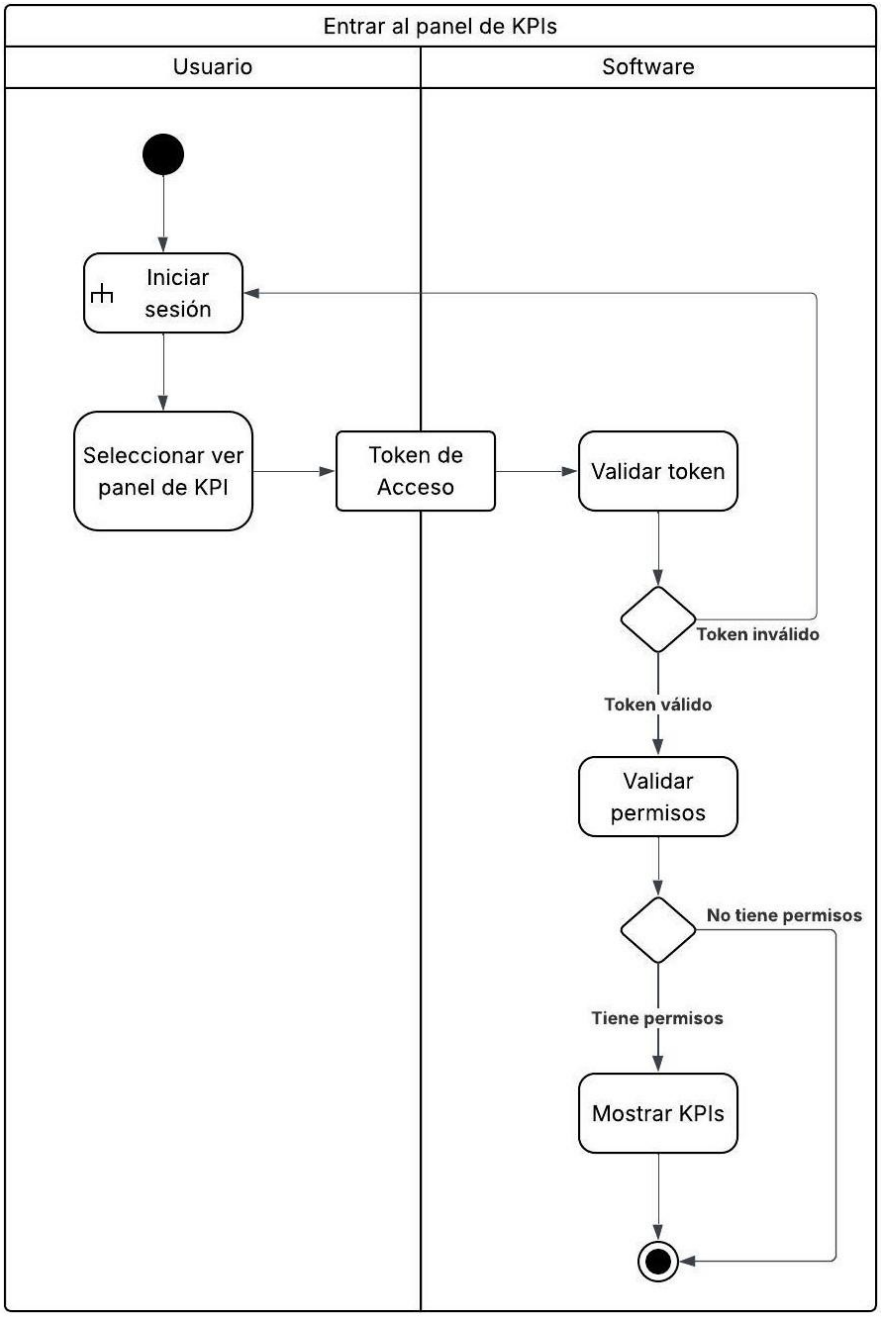


La Figura 16 muestra el flujo de actividades requerido para acceder al panel de indicadores clave de desempeño (KPI) dentro de la plataforma. El proceso se inicia cuando un usuario inicia sesión y selecciona la opción para visualizar el panel de KPI.

El software valida el token de acceso proporcionado y, si es válido, procede a verificar si el usuario cuenta con los permisos necesarios para acceder a dicha funcionalidad. En caso de no contar con los permisos, no se muestra la información. Si los permisos son válidos, el software presenta los indicadores disponibles en el panel.

Figura 16

Diagrama de actividades para entrar al panel de KPIS

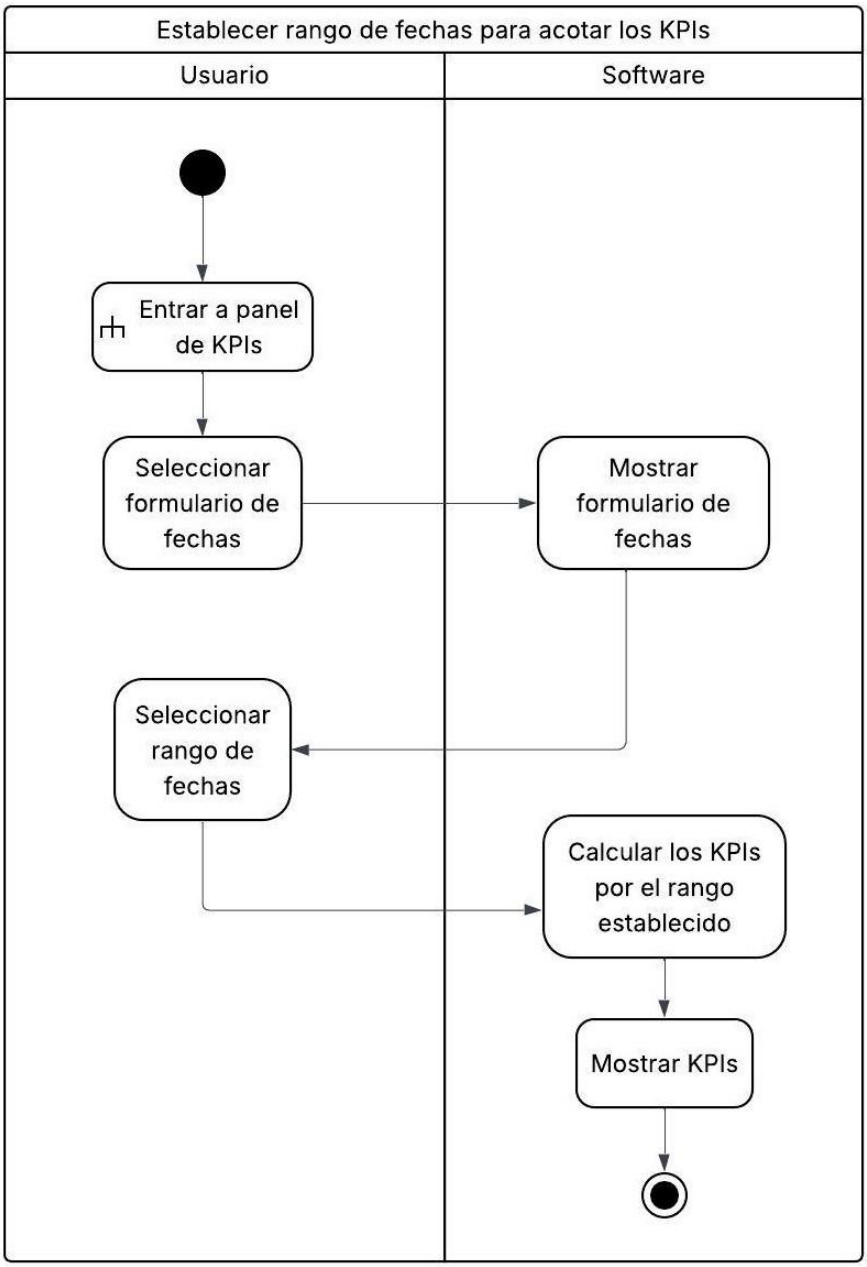


La Figura 17 representa el flujo de actividades asociado a la definición de un rango de fechas para filtrar los indicadores clave de desempeño (KPI) en la plataforma. Este proceso inicia

una vez el usuario ha accedido al panel de KPI, como se describe en la figura 16. Luego, el usuario selecciona la opción para establecer el rango de fechas, tras lo cual el software muestra un formulario destinado a dicha selección. Una vez el usuario define las fechas, el software calcula nuevamente los indicadores con base en el intervalo especificado y actualiza la visualización del panel con los resultados correspondientes.

Figura 17

Diagrama de actividades para ver los KPIs en un rango de fechas



5.3 Arquitectura del Software

La arquitectura del software ha sido diseñada para asegurar que sea escalable, fácil de mantener y que exista una separación definida de responsabilidades. Se basa en un modelo cliente-servidor, donde el cliente es una aplicación web hecha con React y el servidor está formado por una API REST construida en .NET, complementada por un backend en Python (FastAPI) que se encarga de realizar cálculos avanzados y predicciones.

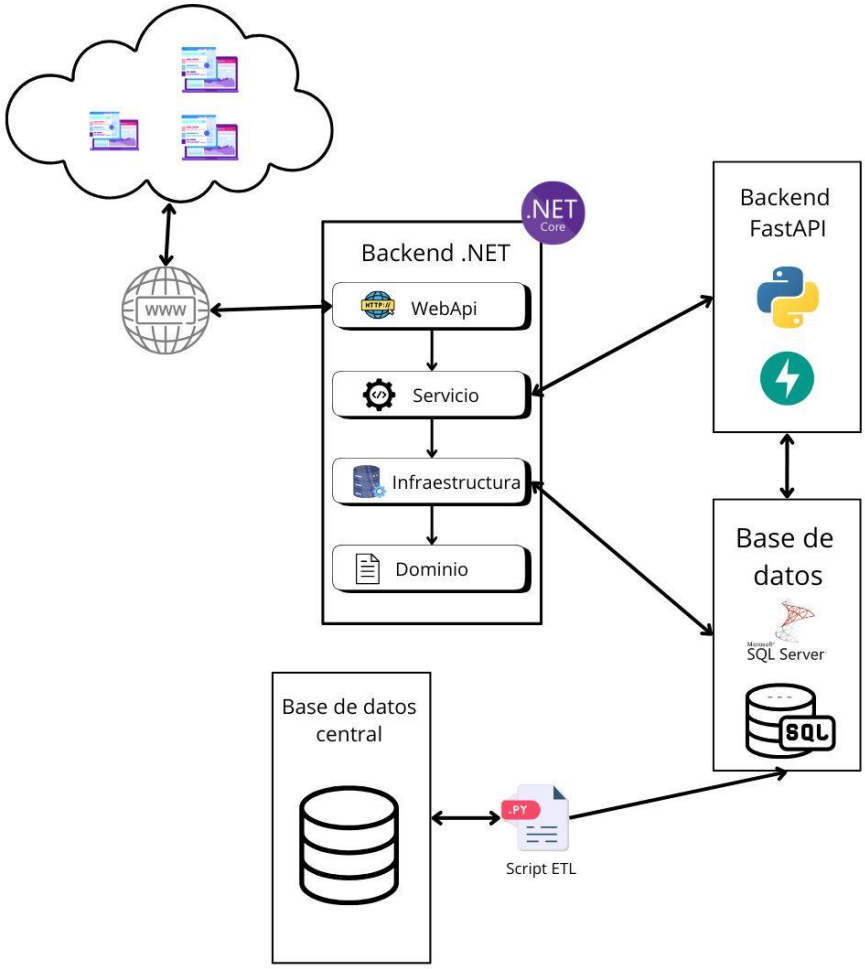
Este enfoque modular y distribuido no solo permite mantener la lógica de negocio y la presentación desacopladas, sino que también facilita la integración de servicios especializados sin poner en riesgo la seguridad ni la coherencia del sistema. La arquitectura fomenta el acceso concurrente desde múltiples dispositivos, favoreciendo la integridad, rendimiento y adaptabilidad a diferentes contextos empresariales.

Además, se han implementado mecanismos de autenticación y autorización robustos basados en JWT, un modelo de multicliente (multitenancy) que optimiza el uso de recursos y simplifica el despliegue, así como un proceso ETL desarrollado en Python que asegura la disponibilidad y consistencia de los datos provenientes de una fuente central. Estos componentes se integran de manera coordinada, formando una solución sólida, extensible y lista para enfrentar futuros requerimientos técnicos y de negocio.

A continuación, se muestra un diagrama que representa la arquitectura general del software, incluyendo los componentes frontend, backend, bases de datos y el proceso ETL.

Figura 18

Diagrama de la arquitectura del software



5.3.1 Arquitectura del Frontend

En el lado del cliente, se ha adoptado una arquitectura híbrida basada en los principios de la arquitectura limpia, integrada al contexto de una aplicación desarrollada con React. Este enfoque combina una organización por capas técnicas compartidas con una modularización por funcionalidades o páginas, lo que permite lograr una alta cohesión, bajo acoplamiento y una excelente escalabilidad.

La arquitectura del frontend se estructura en dos grandes niveles:

1. Capas técnicas compartidas: En la raíz del proyecto se agrupan carpetas que contienen bloques de código reutilizables en toda la aplicación. Estas capas representan la infraestructura común del software y permiten centralizar la lógica relacionada con peticiones HTTP, gestión del estado, validaciones, y componentes de interfaz reutilizables. Este diseño asegura que las funcionalidades puedan desarrollarse de forma desacoplada, reutilizando piezas bien definidas.
2. Módulos funcionales por página: La raíz del proyecto contienen una carpeta pages, la cual agrupa las vistas principales de la plataforma. Cada una de estas páginas se organiza internamente como un módulo funcional completo, con sus propios subcomponentes, servicios, hooks y lógica específica. Esta estructura modular favorece la encapsulación de responsabilidades, ya que cada página administra de manera aislada la lógica que le compete, sin generar dependencias innecesarias con otras partes del software.

Las equivalencias conceptuales entre esta arquitectura y las capas definidas en la arquitectura limpia son las siguientes:

1. Capa de presentación: Corresponde a los componentes visuales y las interfaces de usaurio. Son responsables de mostrar los datos, interactuar con el usuario y emitir eventos hacia la lógica de aplicación.
2. Capa de aplicación: Se representa mediante hooks y funciones específicas que encapsulan el comportamiento de cada página, tales como validaciones, flujos de formularios, o interacción con el estado global. También incluye el enrutamiento y los contextos globales.

3. Capa de infraestructura: Esta capa implementa la comunicación con servicios externos con las APIs del backend, la gestión de sesiones y la administración del estado de la aplicación.
4. Capa de dominio: Es la capa donde definen las estructuras de datos utilizadas en el frontend, ya sean de interfaces globales o respuestas de las peticiones del backend. Estas estructuras están diseñadas para ser independientes de cualquier librería o framework, asegurando su estabilidad y posibilidad de reutilización.

5.3.2 Arquitectura del Backend de Gestión y control

En el lado del servidor, se ha adoptado una arquitectura limpia, que organiza el backend en cuatro capas bien definidas y desacopladas, lo que facilita el desarrollo evolutivo, las pruebas unitarias, la reutilización de componentes y la independencia de tecnologías externas:

1. Web API (Capa de presentación del backend): Esta capa actúa como punto de entrada de todas las solicitudes HTTP desde el cliente. Contiene los controladores y los middlewares, incluyendo el middleware de autenticación y autorización mediante tokens JWT. Aquí se reciben los tokens enviados por los usuarios, se extraen los claims como el ID del usuario, los roles y permisos, y se validan utilizando políticas de autorización configuradas. Esta capa se encarga entonces de verificar si el usuario tiene los permisos necesarios antes de permitir el acceso a cada recurso.
2. Capa de aplicación: Contiene la lógica de negocio de la plataforma. Se encarga de generar y almacenar los tokens JWT, validar su vigencia, gestionar operaciones del software como creación de usuarios, roles y demás, así como de coordinar los flujos entre las demás capas. Además, actúa como puente con servicios externos, como el

backend en Python que calcula indicadores y realiza predicciones. Este diseño centraliza las reglas del negocio, asegurando coherencia y validaciones consistentes.

3. Capa de Infraestructura: En esta capa se encuentran los repositorios que implementan el acceso a la base de datos, utilizando Dapper como micro ORM. Gracias a Dapper, se mantiene un alto control sobre las consultas SQL, logrando eficiencia en el acceso a datos y claridad en la lógica de persistencia.
4. Capa de Dominio: Define las entidades centrales del negocio, representando las estructuras de datos que forman la base del software. Esta capa es completamente independiente del resto de capas y no conoce detalles técnicos como bases de datos o frameworks, lo cual proporciona estabilidad ante posibles cambios tecnológicos.

Este backend también incorpora un mecanismo de autenticación y autorización basado en JWT (JSON Web Tokens) para la autenticación y autorización. Los tokens son generados por la capa de servicio, almacenados en la base de datos, y posteriormente verificados para cada solicitud. Esta estrategia permite mantener sesiones seguras sin necesidad de almacenar el estado del usuario en el servidor.

5.3.3 Arquitectura del Backend para Cálculo de KPIs

El sistema cuenta con un backend complementario desarrollado en Python utilizando el framework FastAPI, cuya función principal es realizar el cálculo de indicadores clave de rendimiento (KPIs) y generar pronósticos de ventas. Este módulo sigue una arquitectura basada en el patrón Modelo-Vista-Controlador (MVC), lo que permite una separación definida entre la lógica de negocio, la estructura de datos y los controladores que gestionan las solicitudes.

La comunicación con este backend se realiza a través de la capa de servicios del backend principal en .NET, lo que permite centralizar el control de acceso y mantener una capa de seguridad consistente en toda la plataforma.

5.3.4 Multicliente

El modelo de multitenencia adoptado permite que una sola instancia de la aplicación y de la base de datos sirva a múltiples empresas. Esta separación lógica de datos se logra mediante el uso de un identificador de empresa presente en las tablas clave de la base de datos. Esta estrategia simplifica la gestión y reduce la complejidad del despliegue, manteniendo la privacidad y separación de la información entre empresas.

5.3.5 Proceso ETL

La plataforma incorpora un proceso ETL (Extract, Transform, Load) desarrollado en Python, diseñado para automatizar la integración y preparación de datos. Este proceso se encarga de extraer información desde una base de datos central, transformarla según los requerimientos del software, y cargarla en la base de datos específica del proyecto, asegurando así la disponibilidad de datos consistentes y útiles para el análisis y predicción.

5.4 Modelo de Base de Datos

La plataforma desarrollada se apoya en dos bases de datos relacionales independientes, diseñadas para cubrir distintos aspectos del funcionamiento del software. La primera base de datos, orientada al control de acceso y gestión de usuarios, estructura la información requerida por los módulos de autenticación, autorización y trazabilidad.

La segunda base de datos almacena los registros históricos de ventas, productos y transacciones comerciales, los cuales son utilizados tanto para la elaboración de prototipos de modelos predictivos como para el cálculo de indicadores clave de rendimiento (KPI). Esta base está estructurada para facilitar consultas analíticas y la agrupación de datos por períodos de tiempo (por ejemplo, ventas diarias o mensuales), operación necesaria para construir series temporales e indicadores agregados.

Ambas bases fueron implementadas en SQL Server, con un diseño enfocado en la normalización, el uso de claves foráneas para preservar la integridad referencial, y la organización modular de las entidades. Esta separación lógica permite aislar las responsabilidades de cada subsistema y facilita tanto el mantenimiento como la evolución futura de la plataforma.

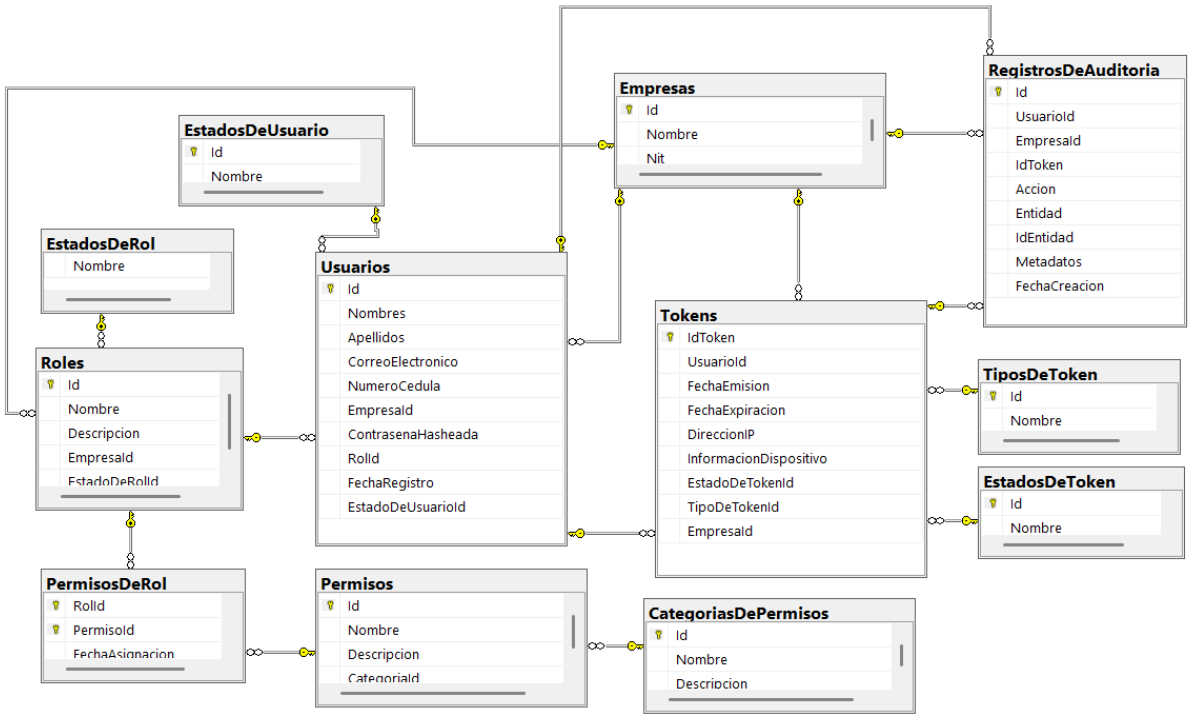
5.4.1 Base de Datos para el Backend de gestión y control

La plataforma cuenta con una base de datos relacional implementada en SQL Server, orientada a respaldar las funcionalidades de autenticación, autorización, gestión de usuarios, roles, permisos y trazabilidad de actividades. La estructura sigue un enfoque multicliente, permitiendo que múltiples empresas compartan una misma instancia de base de datos mediante un mecanismo de aislamiento lógico basado en el campo EmpresaId.

El diseño de esta base de datos se presenta en la Figura 19, donde se representan las relaciones entre las tablas que componen el modelo. Se adoptó una organización modular, estructurando la información en torno a entidades como empresas, usuarios, roles y tokens, con claves foráneas que preservan la integridad referencial del modelo.

Figura 19

Diagrama base de datos gestión y control



La base de datos se compone de doce tablas que representan las entidades del software, las cuales se detallan a continuación:

1. Empresas: almacena los datos básicos de cada organización que utiliza la plataforma. Cada empresa tiene un identificador único (Id), un nombre único (Nombre) y un NIT también único (Nit).
2. EstadosDeRol y EstadosDeUsuario: son tablas auxiliares que representan los distintos estados posibles para los roles y usuarios, respectivamente (por ejemplo, activos, inactivos, eliminados). Estas tablas se relacionan con las entidades Roles y Usuarios mediante claves foráneas, asegurando la normalización de los datos y evitando valores repetitivos.

3. Roles: representa los distintos perfiles de acceso asignados a los usuarios. Cada rol pertenece a una empresa (EmpresaId) y tiene un estado (EstadoDeRolId), lo cual permite una gestión aislada y controlada por organización. A través de la tabla intermedia PermisosDeRol, se definen los permisos asociados a cada rol, permitiendo una estructura flexible y extensible de control de acceso.
4. CategoriasDePermisos y Permisos: estas tablas estructuran los permisos del software, agrupándolos por categorías. La tabla Permisos contiene los derechos de acceso específicos (por ejemplo, crear usuarios), los cuales están organizados mediante claves foráneas hacia CategoriasDePermisos.
5. PermisosDeRol: implementa una relación de muchos a muchos entre Roles y Permisos, permitiendo asignar múltiples permisos a un rol y reutilizar un mismo permiso en diferentes roles. Esta tabla también registra la fecha de asignación mediante el campo FechaAsignacion.
6. Usuarios: representa los individuos registrados en el software. Cada usuario está vinculado a una empresa (EmpresaId), tiene un rol asignado (RolId) y se encuentra en un estado determinado (EstadoDeUsuarioId). Además, se asegura la unicidad del número de cédula y del correo electrónico dentro del ámbito de cada empresa mediante restricciones UNIQUE, lo que impide duplicados en contextos multitenant.
7. TiposDeToken y EstadosDeToken: definen los tipos (por ejemplo, token de acceso, token de recuperación) y estados (activo, expirado, revocado) que puede tener un token de autenticación.
8. Tokens: almacena la información de los tokens generados para autenticación de usuarios. Cada token está vinculado al usuario que lo generó (UsuarioId), a la empresa

correspondiente (EmpresaId), y contiene datos adicionales como dirección IP, dispositivo, fecha de emisión y expiración, estado y tipo del token.

9. RegistrosDeAuditoria: es una tabla fundamental para la trazabilidad de las acciones realizadas en el software. Registra quién realizó qué acción, sobre qué entidad, en qué momento, y desde qué token o empresa, en caso de que aplique. Los campos clave (UsuarioId, EmpresaId, IdToken) permiten nulos para admitir registros globales o externos, sin sacrificar la integridad referencial cuando sí existan los vínculos.

Todas las relaciones entre tablas se implementan mediante claves foráneas, lo que permite mantener la integridad referencial y realizar consultas relacionales robustas. Además, se utilizaron restricciones UNIQUE para asegurar la unicidad de atributos relevantes como correos, NIT, nombres de entidades, etc.

La tabla 25 es un resumen de las entidades de la base de datos y su propósito en la plataforma:

Tabla 25

Resumen entidades de la base de datos de gestión y control

Tabla	Propósito
Empresas	Almacena la información de las empresas que utilizan el software.
EstadosDeRol	Define los posibles estados de un rol (por ejemplo, activo o eliminado).
Roles	Representa los distintos roles que pueden tener los usuarios dentro de una empresa, con su estado correspondiente.
CategoriasDePermisos	Agrupa los permisos según su categoría funcional (por ejemplo: usuarios, ventas, auditoría, etc.).
Permisos	Define permisos específicos que se pueden asignar a los roles para controlar el acceso a funcionalidades.

PermisosDeRol	Relaciona roles con permisos específicos y almacena la fecha de asignación.
EstadosDeUsuario	Indica el estado de los usuarios (por ejemplo: activo, deshabilitado, no confirmado).
Usuarios	Almacena los datos de los usuarios del software, asociados a roles, empresas y estados.
TiposDeToken	Define los tipos de token utilizados en el software (por ejemplo: acceso, recuperación de contraseña, etc.).
EstadosDeToken	Registra los posibles estados de los tokens (por ejemplo: activo, revocado, expirado).
Tokens	Gestiona la autenticación y sesión de los usuarios, incluyendo información del dispositivo, IP y tiempos de emisión y expiración.
RegistrosDeAuditoria	Almacena eventos relevantes del software, incluyendo la acción realizada, entidad afectada, usuario, token, empresa y metadatos asociados.

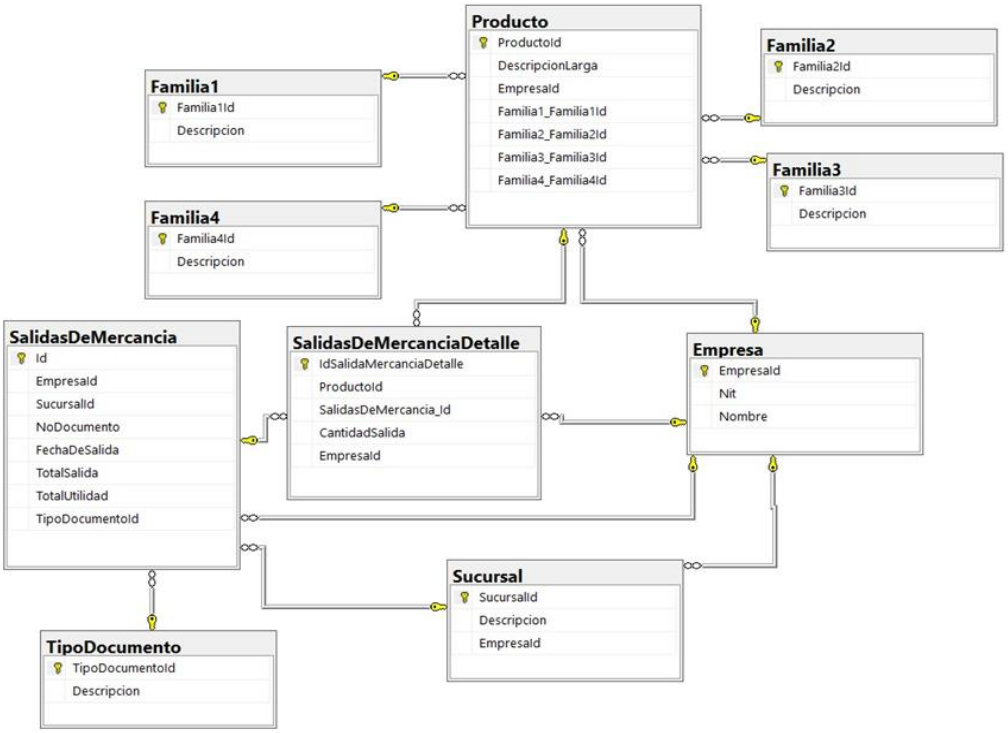
5.4.2 Base de datos para indicadores y prototipos de modelos predictivos

La plataforma incorpora una base de datos relacional orientada al almacenamiento de información operativa relacionada con ventas, productos y transacciones comerciales. Esta estructura es utilizada en los procesos de cálculo de indicadores clave de desempeño (KPI) y en la construcción de prototipos de modelos predictivos basados en series temporales.

El modelo de datos se diseñó sobre SQL Server y permite representar relaciones entre entidades como empresas, sucursales, productos, documentos de salida y sus detalles, incluyendo además clasificaciones jerárquicas del portafolio de productos. La Figura 20 presenta el esquema de esta base de datos, donde se observa la relación entre las tablas que permiten organizar cronológicamente los datos requeridos para análisis históricos y agregaciones temporales.

Figura 20

Base de datos para los KPIS



Nota. Modelo relacional de base de datos diseñado para el software de gestión de salidas de mercancía. Incluye entidades como *Producto*, *Empresa*, *Sucursal*, *SalidasDeMercancia* y tablas auxiliares relacionadas con clasificación jerárquica del producto y tipos de documento. Las relaciones se representan mediante claves primarias y foráneas.

A continuación, se describen las tablas que conforman este modelo:

1. Producto: Tabla que almacena los datos principales de los productos, incluyendo su identificador único (ProductoId), descripción y las referencias a las categorías jerárquicas (Familia1, Familia2, Familia3, Familia4) que permiten una clasificación detallada.

2. Familia1, Familia2, Familia3, Familia4: Representan niveles de categorización del producto. Cada familia tiene su propia tabla con un identificador y descripción, lo que permite una estructuración jerárquica y flexible de los productos.
3. Empresa: Contiene información básica de cada empresa, como su identificador (EmpresaId), nombre y número de identificación tributaria (NIT). Esta entidad se relaciona con varias otras tablas clave del modelo.
4. Sucursal: Tabla que registra las sucursales asociadas a cada empresa. Incluye un identificador único, una descripción y la clave foránea hacia la tabla Empresa.
5. TipoDocumento: Contiene la codificación y descripción de los tipos de documentos usados en las transacciones de salida de mercancía.
6. SalidasDeMercancia: Registra cada transacción de salida, incluyendo datos como fecha, total de la transacción, utilidad, empresa, sucursal y tipo de documento.
7. SalidasDeMercanciaDetalle: Registra el detalle de cada salida, incluyendo el producto despachado, la cantidad de unidades y la empresa correspondiente. Esta tabla se relaciona directamente con Producto, Empresa y SalidasDeMercancia.

El modelo relacional implementado en la plataforma cuenta con múltiples relaciones entre entidades que permiten mantener la integridad de los datos y optimizan los procesos de consulta. A continuación, se detallan las conexiones más relevantes del modelo:

1. Producto se relaciona con la entidad Empresa a través del campo EmpresaId, lo que permite identificar a qué empresa pertenece cada producto. Además, se conecta con cuatro niveles de familias jerárquicas:

   Familia1_Familia1Id → Familia1,

   Familia2_Familia2Id → Familia2,

Familia3_Familia3Id → Familia3,

Familia4_Familia4Id → Familia4.

Estas relaciones permiten una clasificación detallada de los productos y facilitan su análisis desde diferentes perspectivas jerárquicas.

2. SalidasDeMercancia se conecta con:

- Empresa, mediante EmpresaId,
- Sucursal, mediante SucursalId,
- TipoDocumento, mediante TipoDocumentoId,
- Además, mantiene una relación uno-a-muchos con la entidad SalidasDeMercanciaDetalle, a través del campo Id, que corresponde al campo SalidasDeMercancia_Id en la tabla detalle.

3. SalidasDeMercanciaDetalle representa el desglose de cada salida y contiene:

- Una relación con Producto mediante ProductoId,
- Una relación con Empresa mediante EmpresaId,
- Una relación con SalidasDeMercancia mediante SalidasDeMercancia_Id.

4. Sucursal también se relaciona con Empresa mediante el campo EmpresaId, indicando a qué empresa pertenece cada sede o punto de venta.

Estas conexiones reflejan un diseño normalizado que facilita el mantenimiento del software, la ejecución de consultas eficientes y la extracción de información precisa para la toma de decisiones empresariales. El uso correcto de claves primarias y foráneas asegura la consistencia de los datos y habilitas operaciones analíticas a gran escala.

5.5 Implementación de los indicadores clave de rendimiento (KPI)

Esta sección presenta la implementación técnica y funcional de la plataforma web desarrollada, detallando los principales componentes que la integran y la forma en que se articulan para alcanzar los objetivos del proyecto. Se abordan cuatro ejes fundamentales: el desarrollo del frontend, centrado en la experiencia del usuario y la presentación visual de los datos; el proceso ETL (extracción, transformación y carga), responsable de consolidar la información desde fuentes operativas; la construcción del backend, encargado de la lógica de negocio, la gestión de usuarios y la exposición de servicios mediante API; y, finalmente, la integración de modelos de predicción basados en los datos consolidados. A lo largo de esta sección se describen las funcionalidades implementadas, las tecnologías seleccionadas y las decisiones arquitectónicas adoptadas para asegurar el funcionamiento eficiente, seguro y escalable de la plataforma.

5.5.1 Frontend (interfaces de usuario)

La aplicación se desarrolló siguiendo principios de diseño minimalista y funcional, con el objetivo de asegurar una operación ágil, una visualización comprensible de la información y una navegación sencilla para los usuarios.

5.5.1.1 Estructura de la aplicación

Con el fin de lograr una separación definida de responsabilidades, una mejor organización del código fuente, y una base sólida para realizar pruebas, para mantener la aplicación y poder escalarla a futuro, se optó por una estructura basada en los conceptos de arquitectura limpia, donde cada carpeta pertenece a una de las capas de la arquitectura mencionada:

1. Components: Carpeta que contiene los componentes reutilizables de la aplicación.

2. Context: Encargado del manejo del estado general de la aplicación junto a los interceptores.
3. Guards: Encargados de la constante validación del enrutamiento de la aplicación.
4. Hooks: Contiene bloques de código reutilizables para los componentes de React.
5. Models: Continene las interfaces y estructuras de datos que se utilizaran en la aplicación.
6. Services: Contiene las comunicaciones con servicios externos como llamadas a APIs.
7. Utilities: Contiene funciones generales que pueden ser reutilizables tanto en componentes de React como en otros bloques de código.
8. Pages: Contienen las vistas principales de la aplicación, dentro de cada página, se contienen una estructura similar a la de la raíz, correspondiente a las funcionalidades que únicamente se utilizan en ese modulo.
9. __test__: Contienen las pruebas de la aplicación.

Las siguientes figuras dan una representación visual de la estructura mencionada:

Figura 21

Estructura del frontend

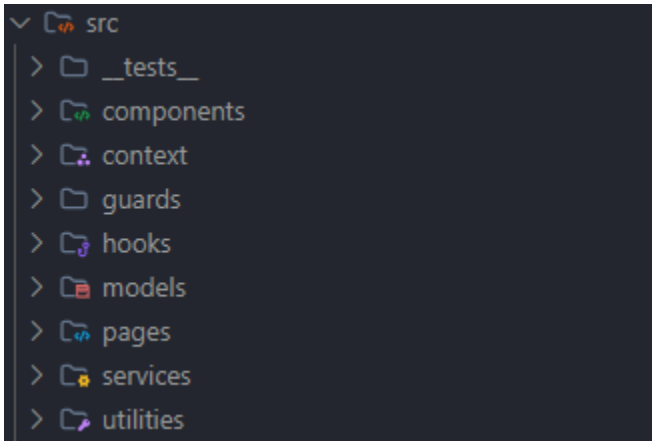
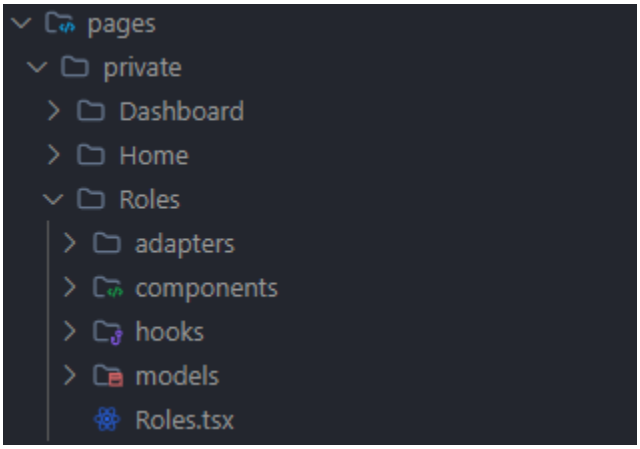


Figura 22

Estructura del frontend - Módulo específico



5.5.1.2 Selección de colores

Para el desarrollo visual de la aplicación, se implementó un sistema de colores basado en variables CSS personalizadas integradas con Tailwind CSS, siguiendo la convención propuesta por el sistema de diseño Shadcn/ui. Este enfoque permite mantener consistencia visual, accesibilidad y soporte nativo para temas claro y oscuro.

Los colores fueron definidos en formato HSL (Hue, Saturation, Lightness), lo que facilita su manipulación programática y permite mantener una buena relación de contraste. Las variables se agrupan por contexto de uso y modo visual.

En la siguiente figura se presenta un resumen visual de la paleta de colores utilizada en el proyecto.

Figura 23

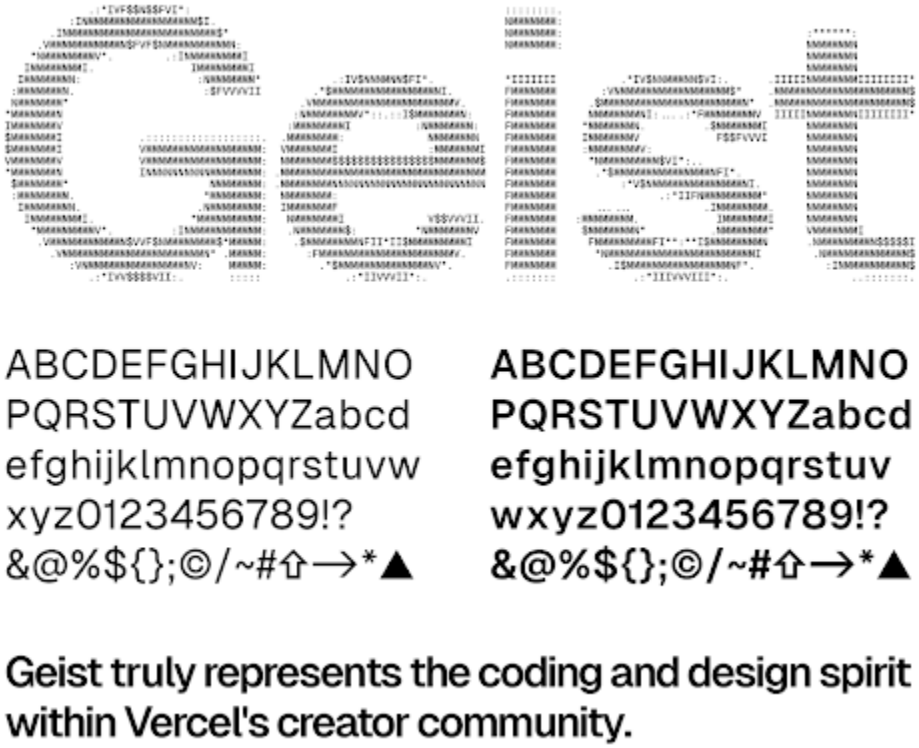
Paleta de colores principales de la aplicación

5.5.1.3 Tipografía

Para el diseño tipográfico del proyecto se seleccionó la familia Geist de Google Fonts. Esta elección responde a su estética limpia y contemporánea, ideal para aplicaciones con una interfaz moderna, así como a su excelente legibilidad en pantalla. Geist, al ser una tipografía sans-serif sobria y versátil, permite mantener una jerarquía visual estructurada y coherente en cada sección de la interfaz, desde los títulos hasta el contenido principal.

Figura 24

Representación visual de la fuente tipográfica Geist



Nota. Tomado de *Geist,* por Google Fonts, 2025, https://fonts.google.com/specimen/Geist/about

5.5.1.4 Iconografía

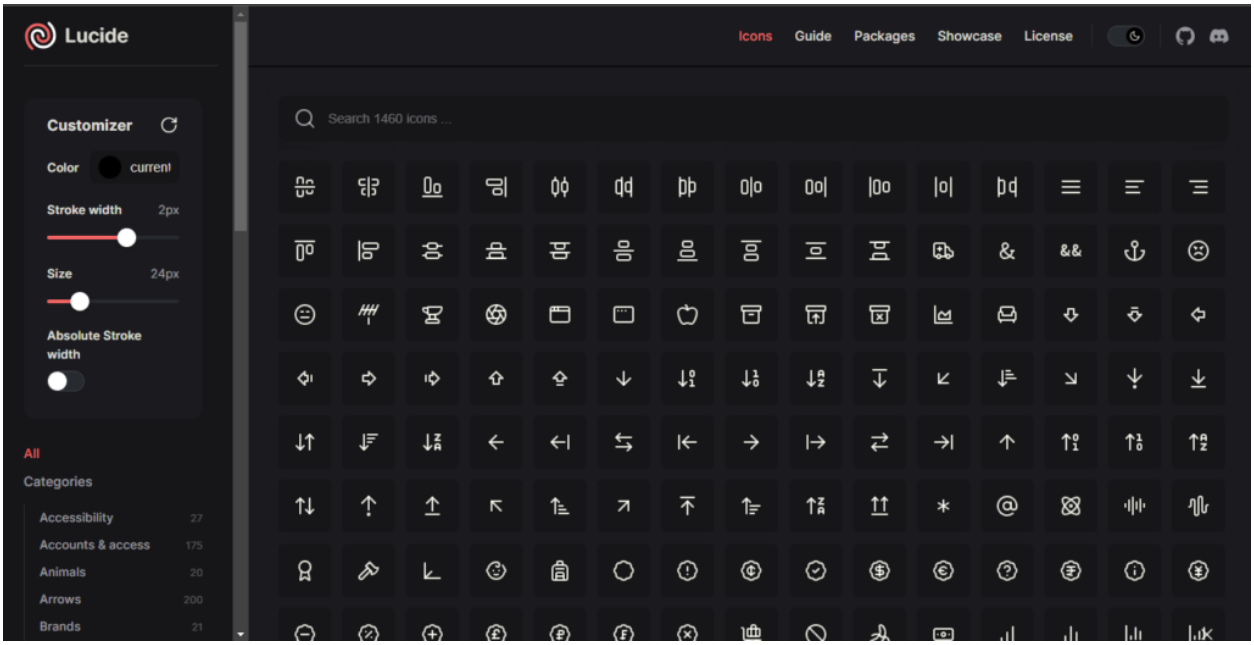
Para la iconografía del proyecto, se seleccionaron los iconos de Lucide React, los cuales son los predeterminados en la librería Shadcn. Esta elección se fundamenta en su estilo minimalista y moderno, que complementa la estética limpia y contemporánea de la tipografía Geist.

Al ser vectoriales y escalables, los iconos de Lucide React permiten una visualización nítida y coherente en distintos tamaños y resoluciones, mejorando así la experiencia del usuario.

Además, su integración sencilla con React facilita el desarrollo y mantenimiento de la interfaz, asegurando consistencia visual y funcional en los elementos gráficos del proyecto.

Figura 25

Conjunto de iconos Lucide React empleados en la interfaz de la plataforma



Nota. Captura de pantalla. Tomado de Home, por Lucide, s.f., https://lucide.dev/icons/

5.5.1.5 Vistas principales

Las siguientes capturas corresponden a las pantallas completas de la plataforma en su versión de escritorio, donde se puede observar la organización visual, jerarquía de información y aplicación del diseño.

Figura 26

Vista de inicio de sesión

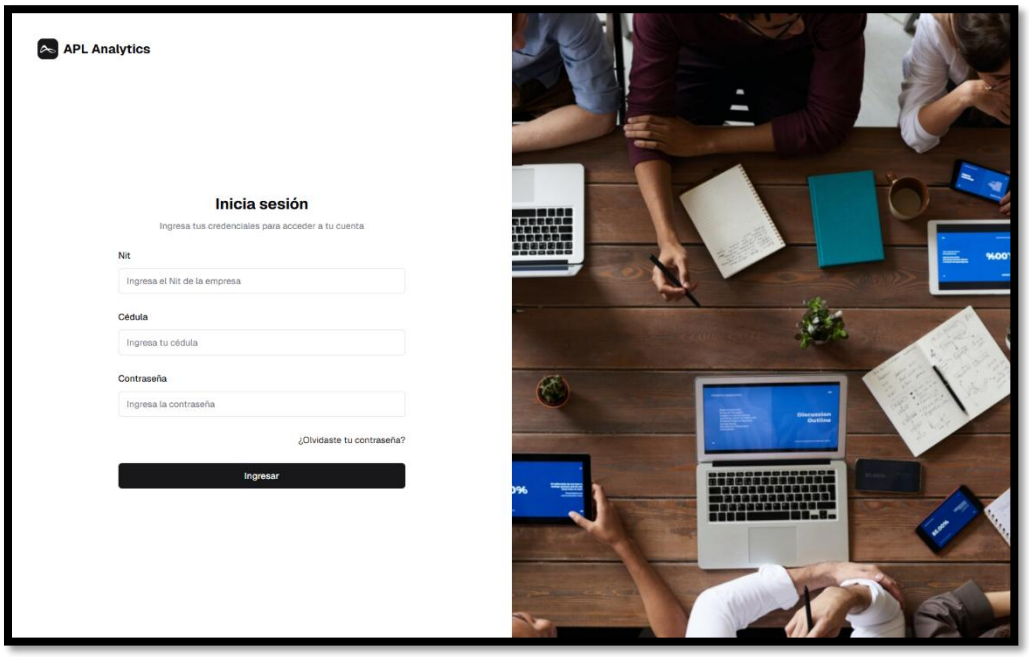


Figura 27

Vista de recuperación de contraseña

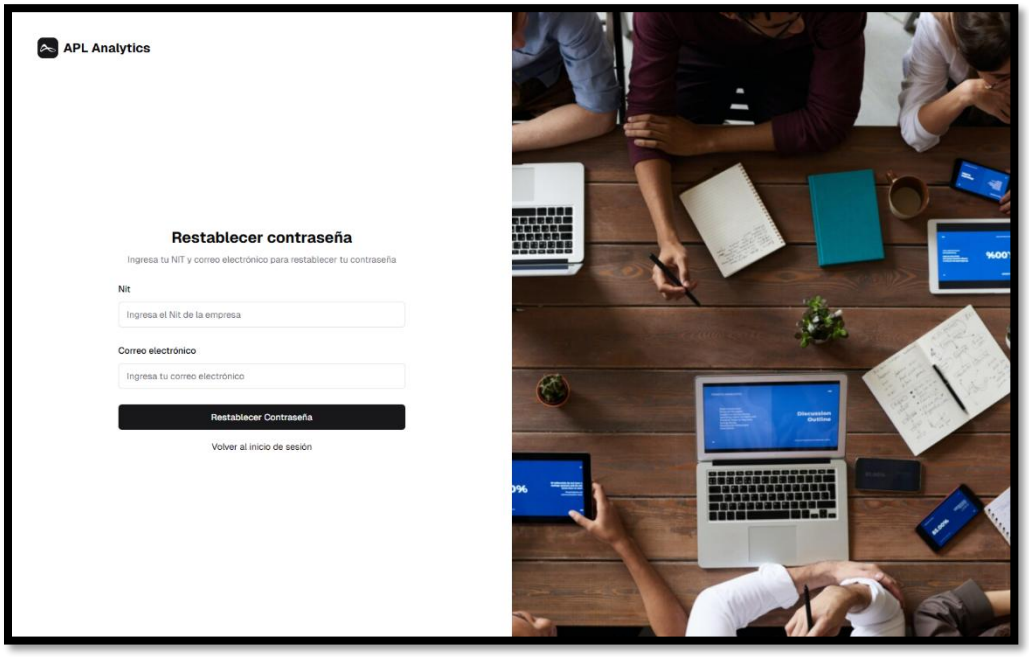


Figura 28

Vista de ingreso de contraseña nueva

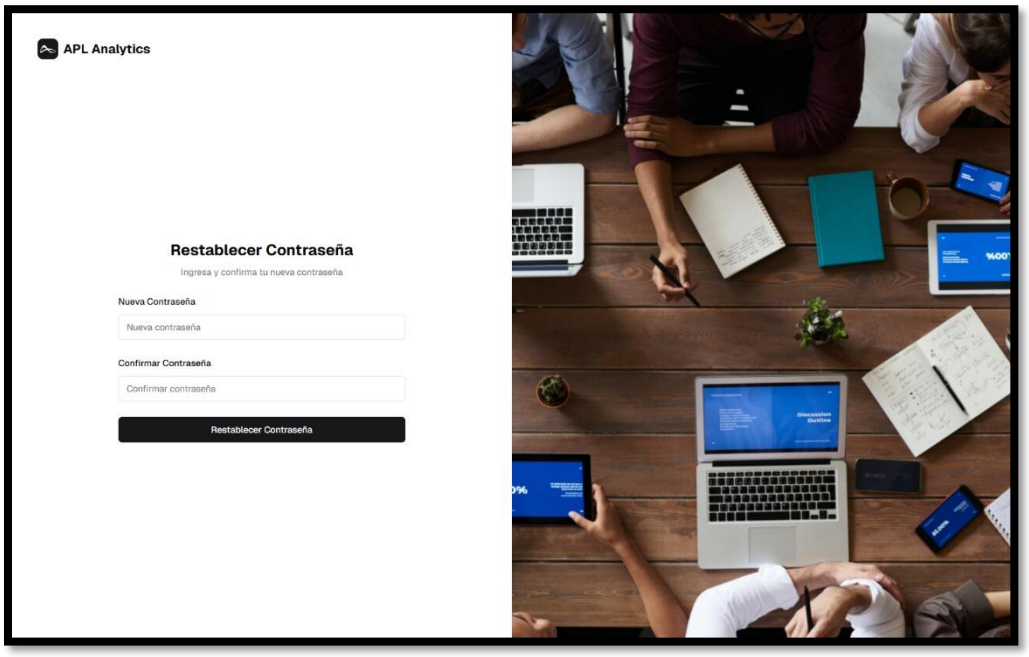


Figura 29

Vista de cuenta verificada correctamente

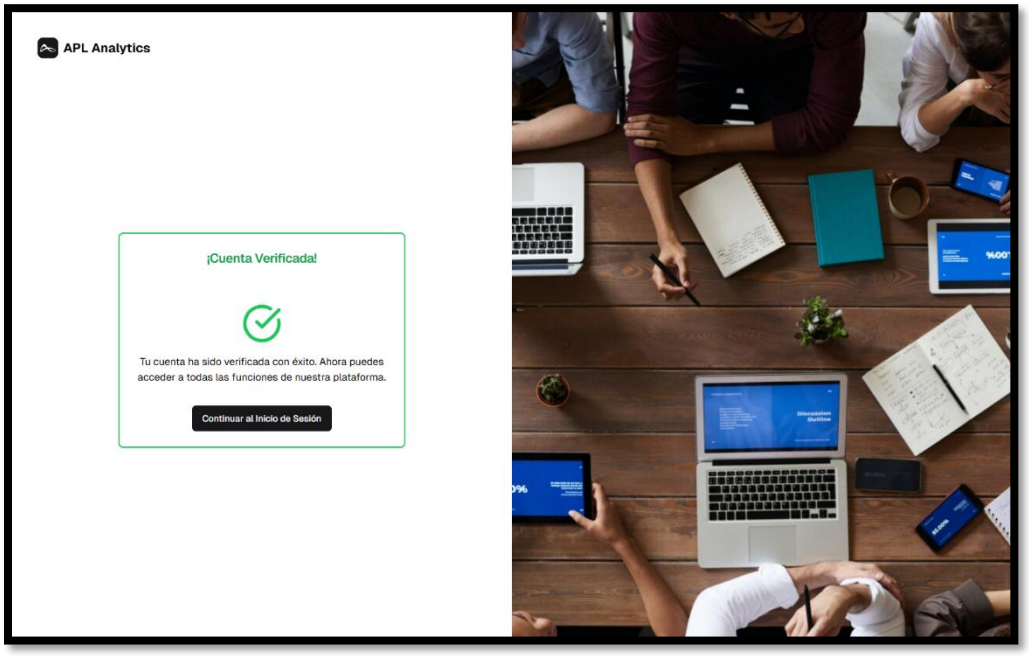


Figura 30

Vista de carga de verificación de cuenta

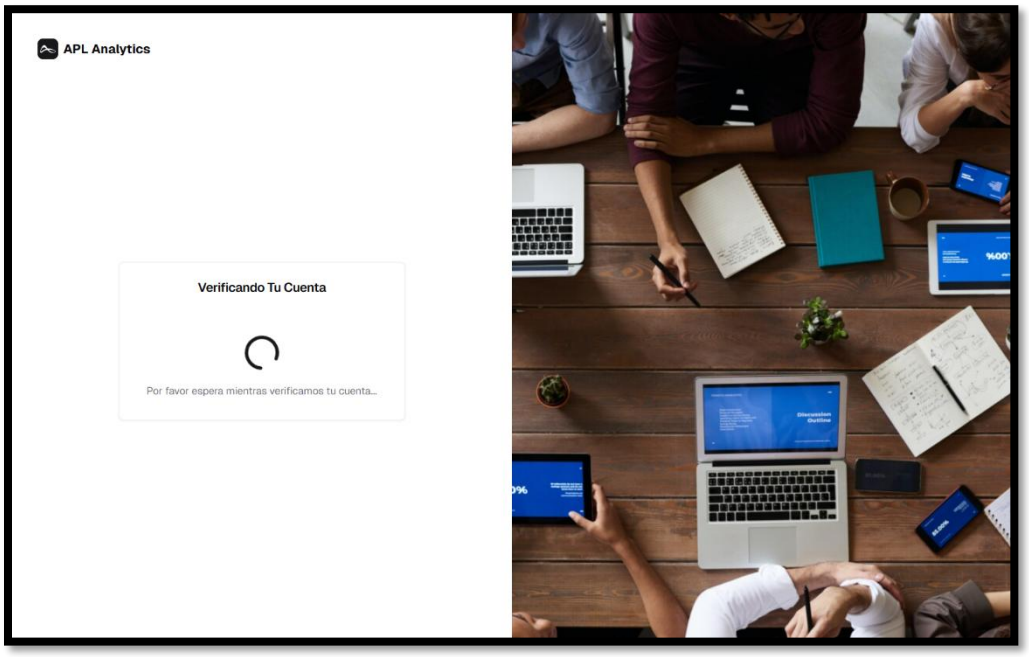


Figura 31

Vista de verificación de cuenta fallida

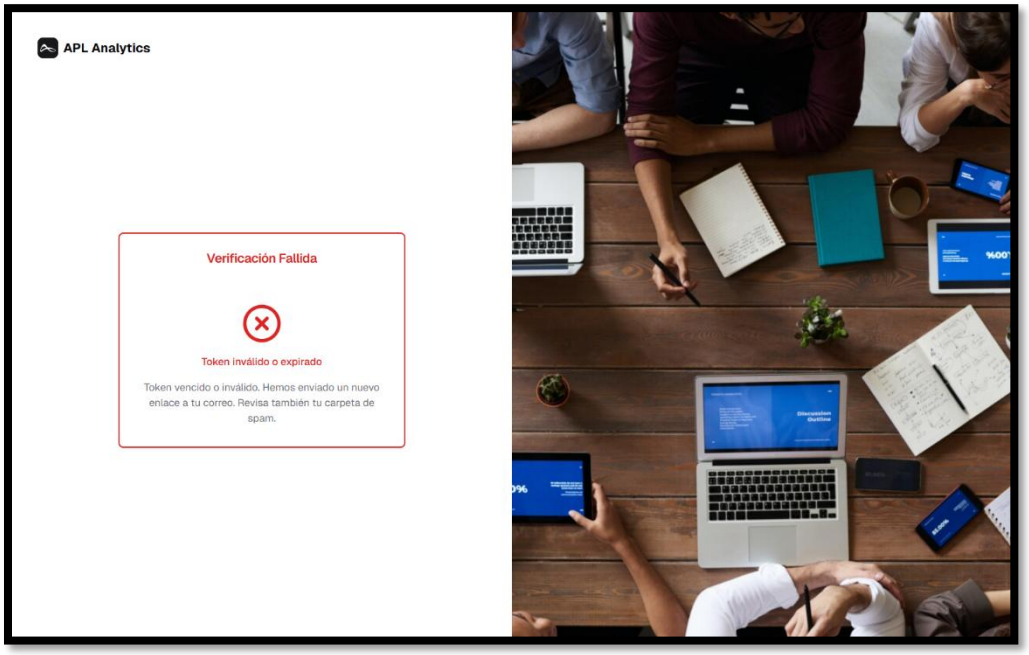


Figura 32

Vista de inicio

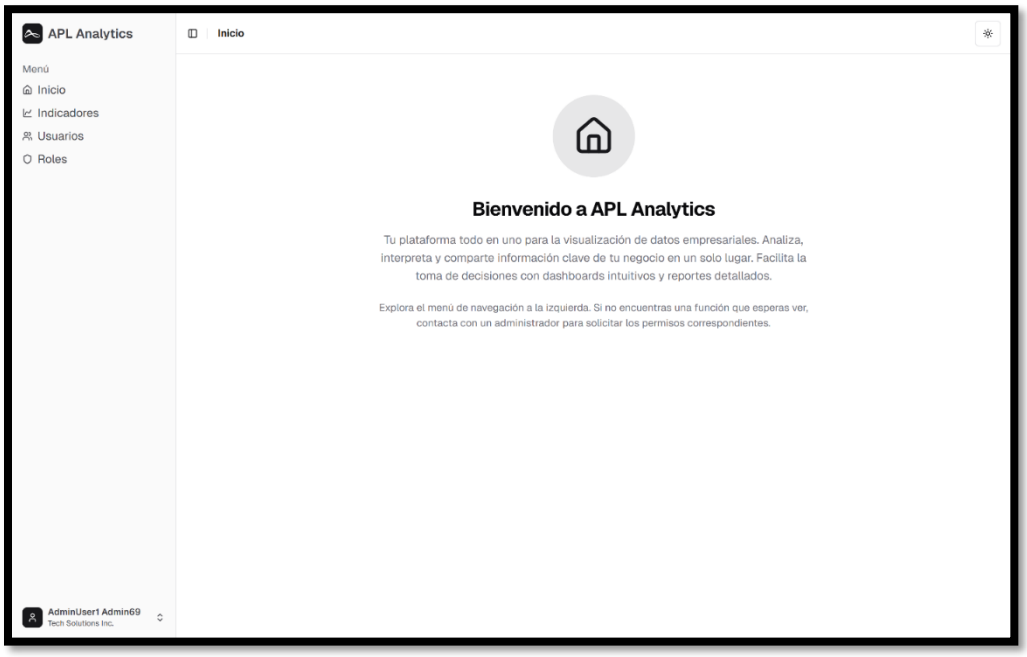


Figura 33

Vista de indicadores de desempeño

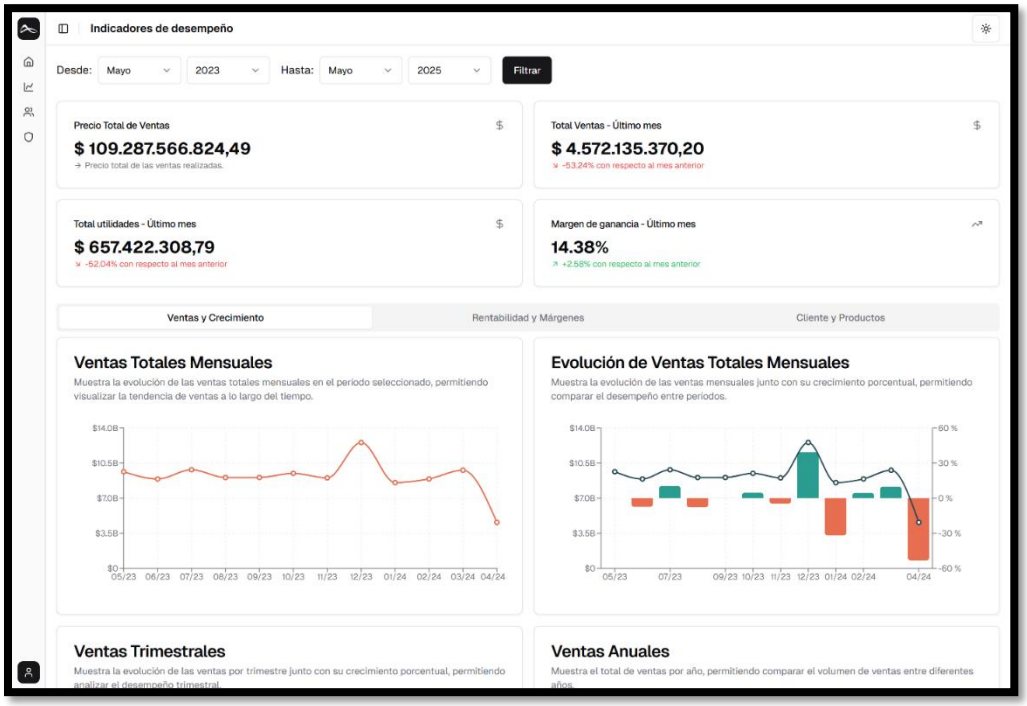


Figura 34

Vista de gestión de usuarios

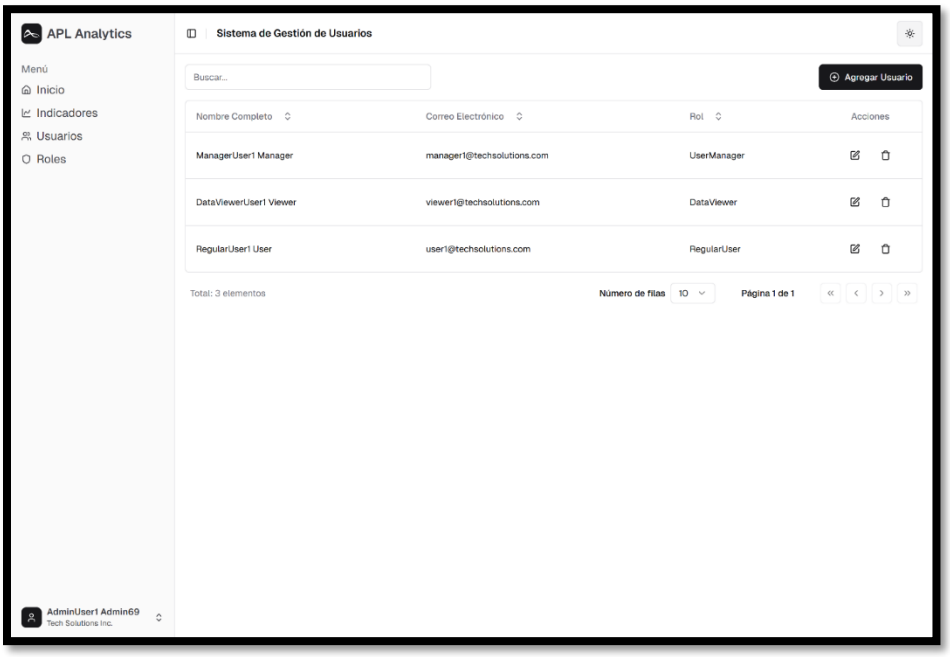


Figura 35

Vista de creación de usuario

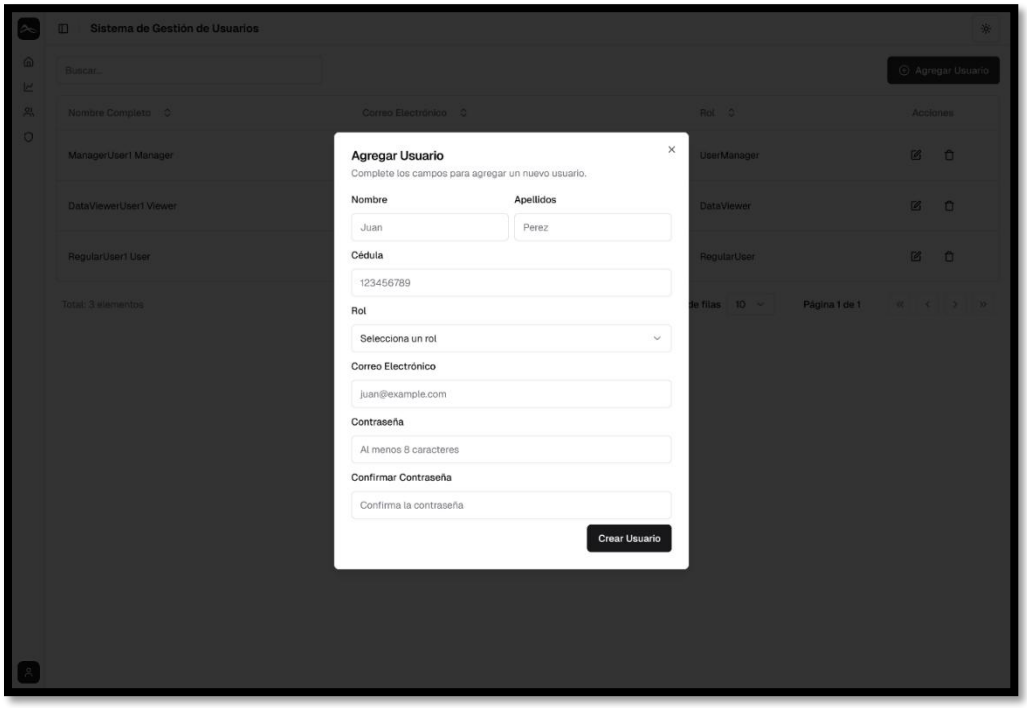


Figura 36

Vista de edición de usuarios

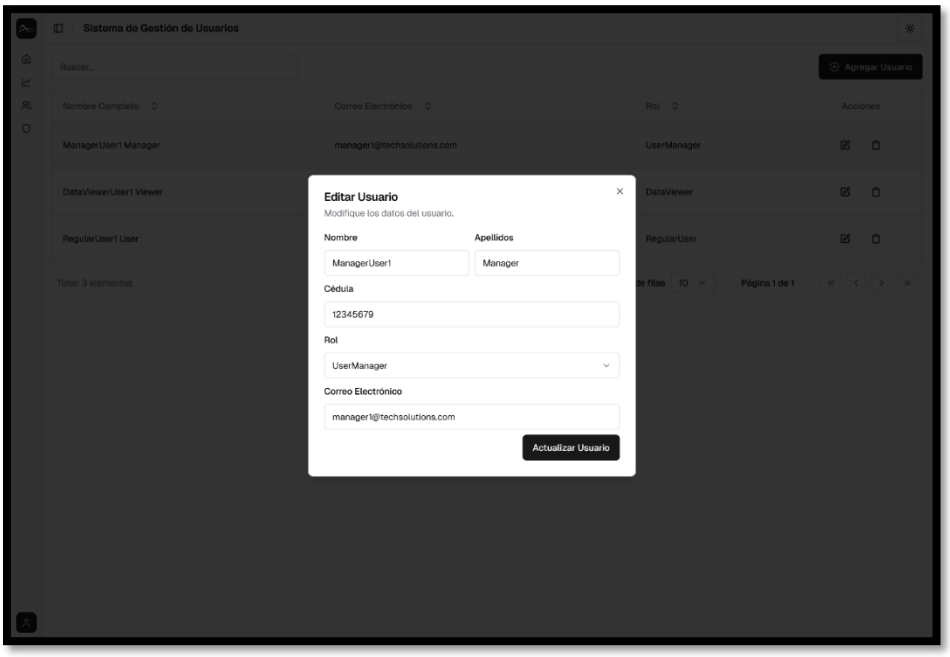


Figura 37

Vista de eliminación de usuario

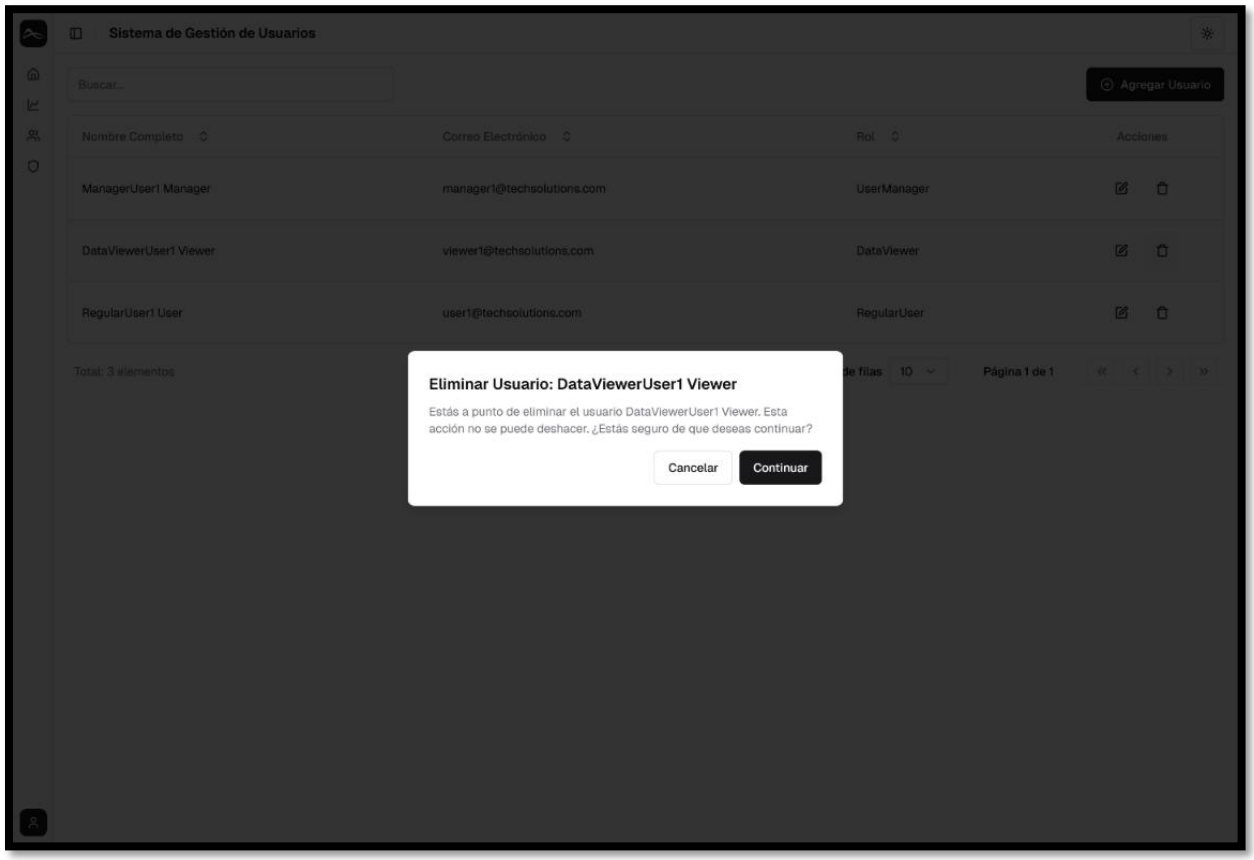


Figura 38

Vista de gestión de roles

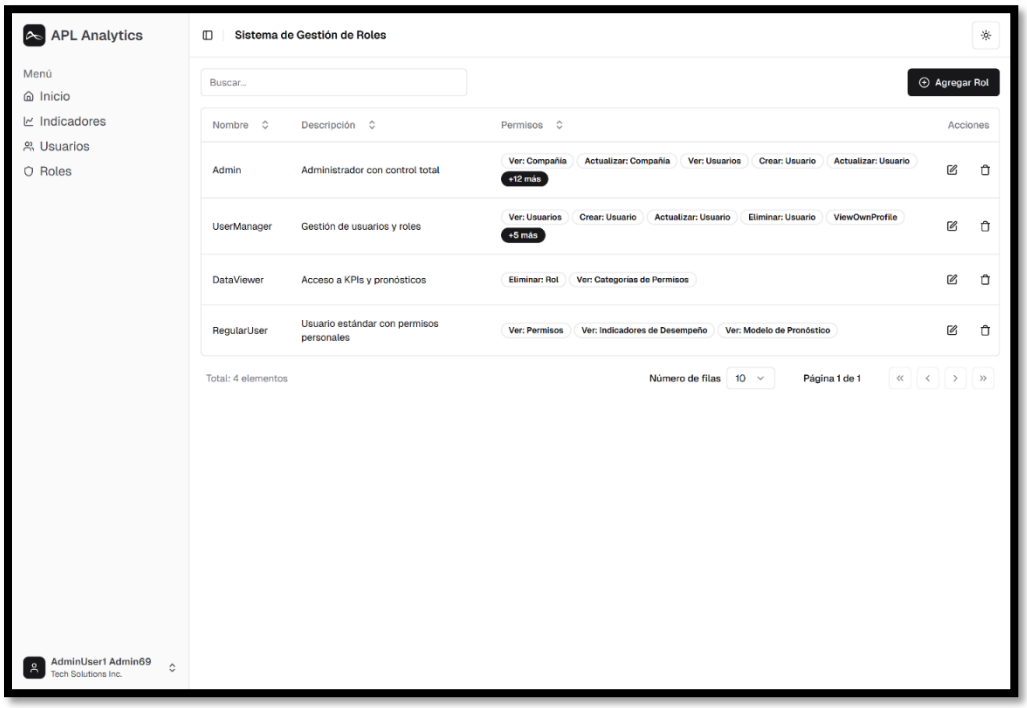


Figura 39

Vista de creación de rol

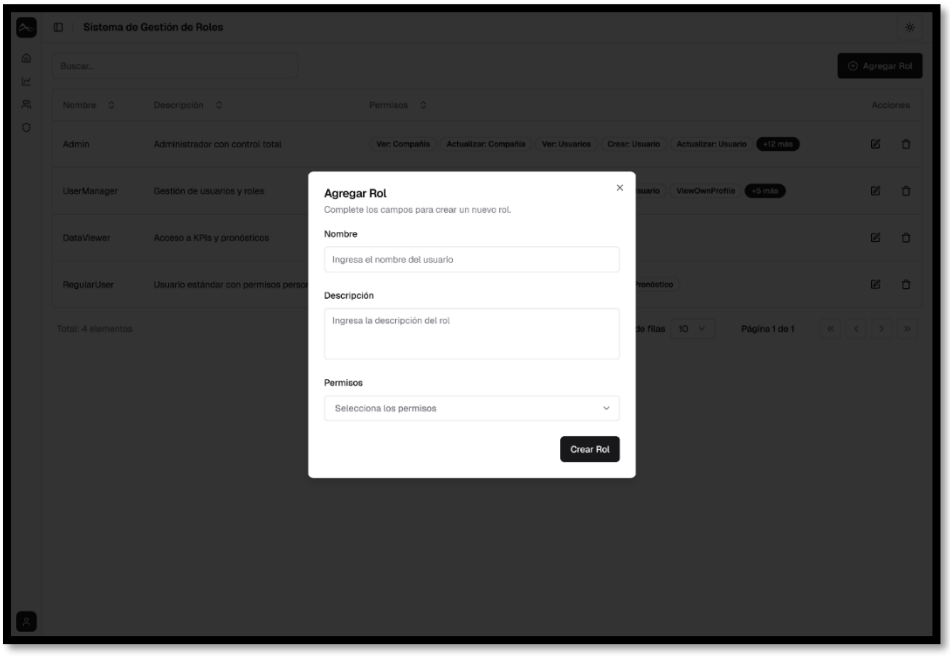


Figura 40

Vista de edición de rol

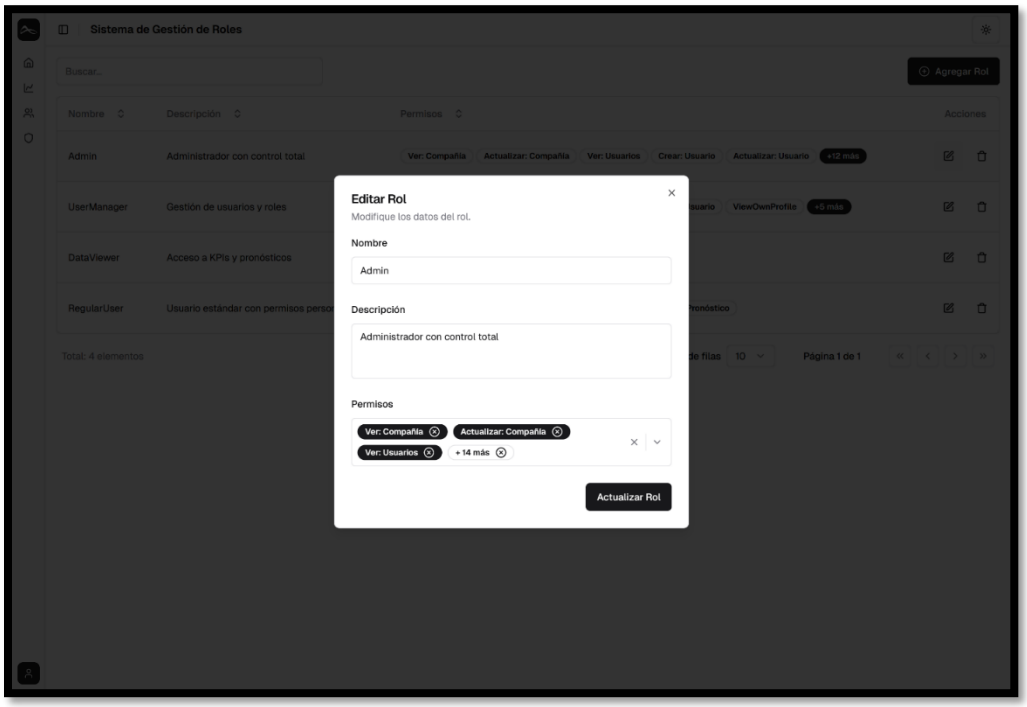
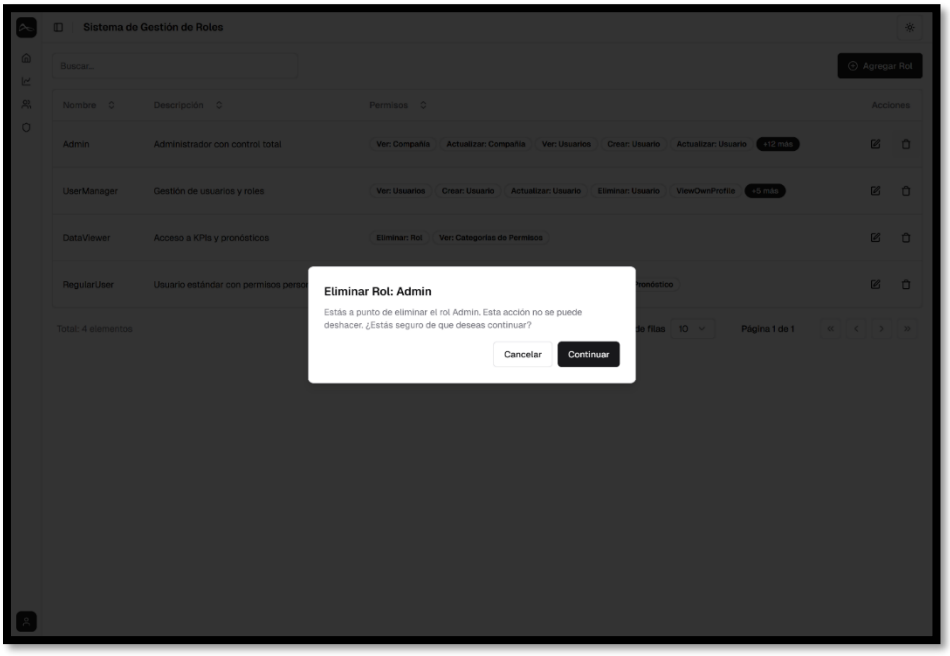


Figura 41

Vista de eliminación de rol



5.5.1.6 Componentes reutilizables

Las siguientes capturas muestran componentes clave de la plataforma utilizados de forma repetida en distintas vistas, evidenciando su consistencia visual y funcional:

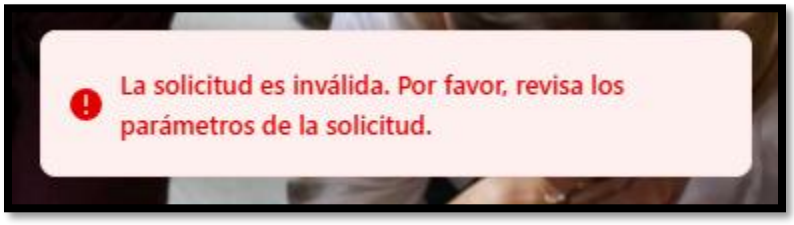
1. Snackbar (sooner): Componente de notificación breve que aparece en la parte superior derecha de la pantalla para informar al usuario sobre acciones realizadas o eventos del software. Su diseño es discreto pero visible, y utiliza el color como indicador para comunicar si la acción fue exitosa o no.

Figura 42

Sooner notificando una operación exitosa

Figura 43

Sooner notificando una operación fallida



2. Menú lateral: Elemento de navegación fijo que permite el acceso rápido a las distintas secciones de la aplicación a las que el usuario tenga permiso de acceso, así como la realización de funcionalidades relacionadas con la sesión activa, como la edición del perfil o el cambio de contraseña. Su diseño jerárquico y compacto facilita la

orientación, mientras que su comportamiento responsivo permite una experiencia fluida en pantallas de diversos tamaños.

Figura 44

Menú lateral

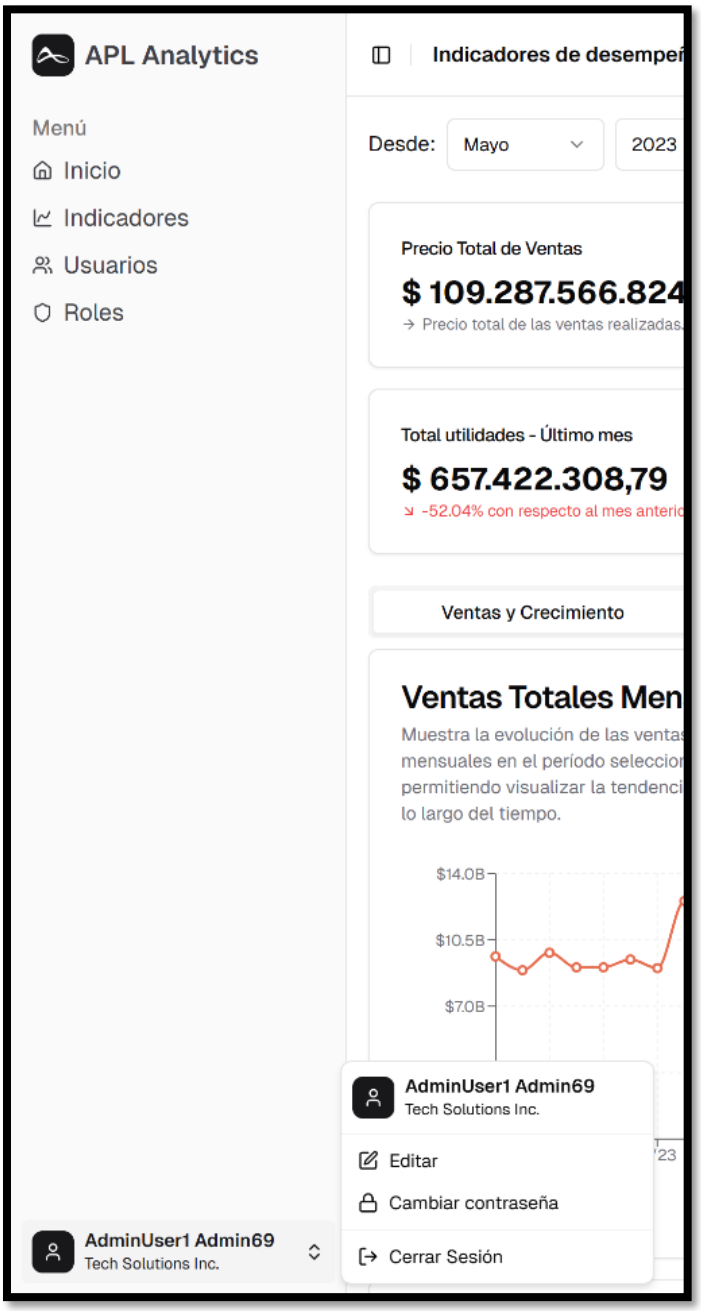


Figura 45

Vista de cambio de contraseña del perfil

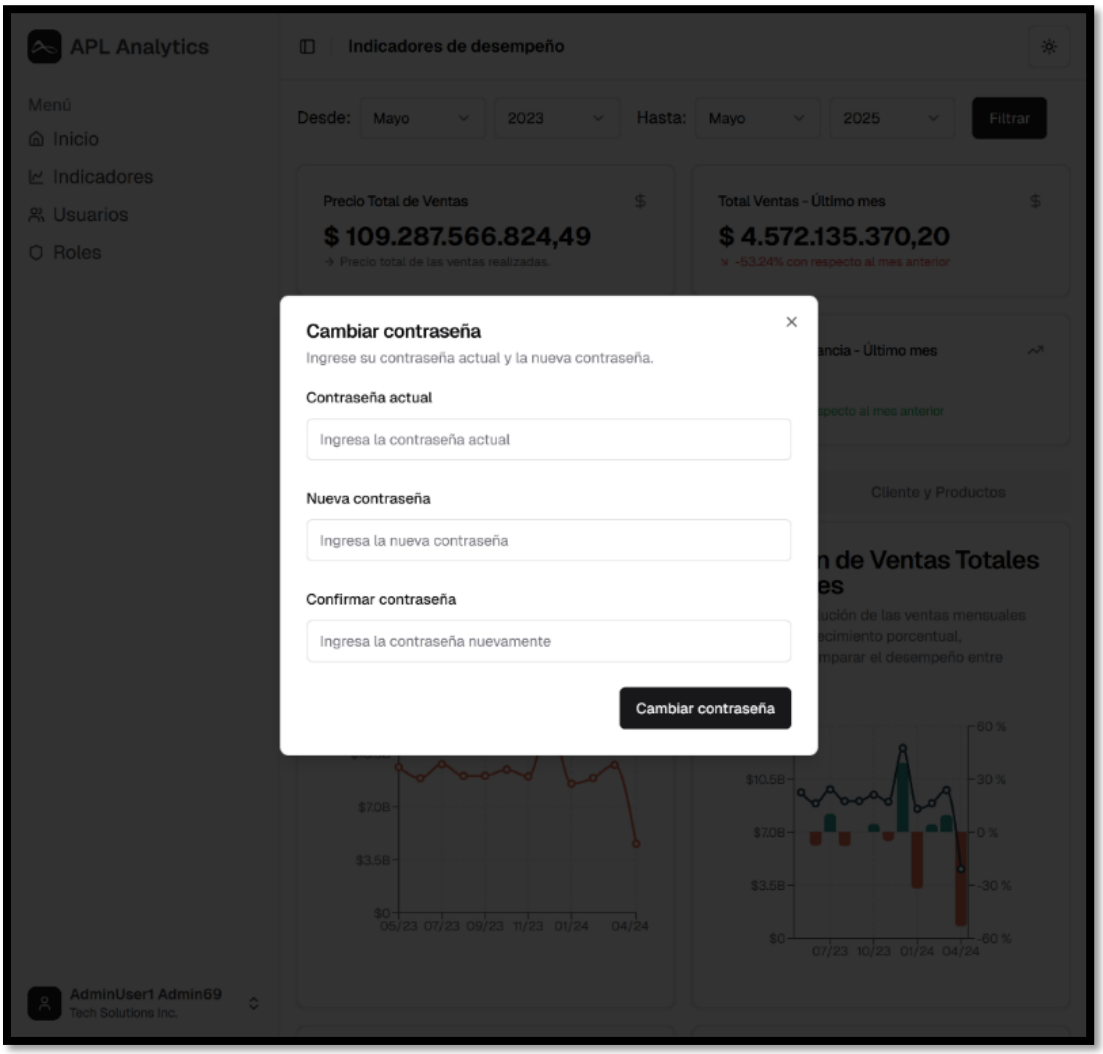
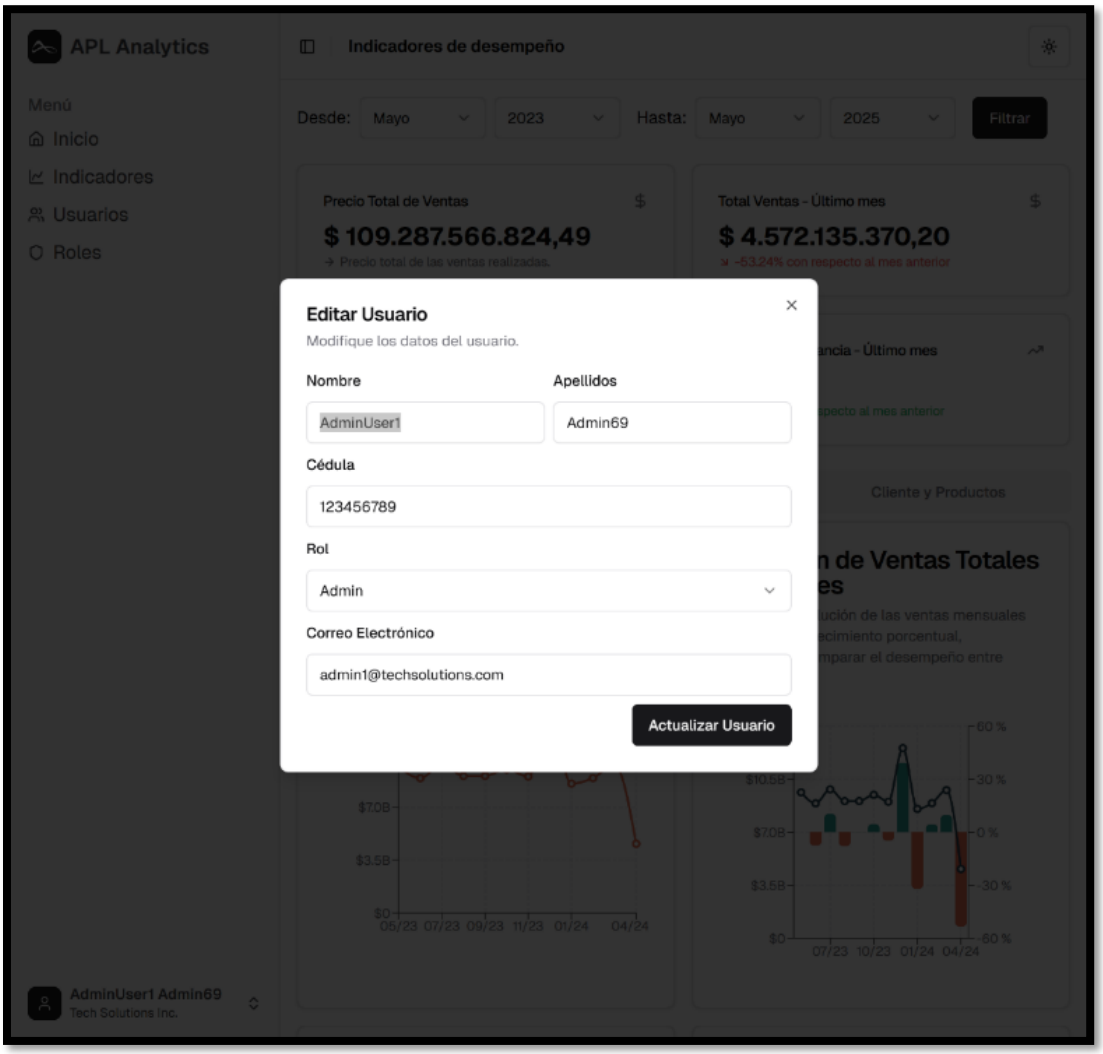


Figura 46

Vista de edición de perfil



3. Indicadores de un solo valor: Pequeños bloques informativos que resumen métricas clave, como totales, promedios o cambios porcentuales. Están diseñados para destacar información puntual con claridad y, por lo general, se acompañan de un ícono o etiqueta descriptiva, así como de información comparativa respecto a su valor anterior.

Figura 47

Indicador de valor único neutro

Figura 48

Indicador de valor único decreciente

Figura 49

Indicador de valor único creciente

Figura 50

Indicador de valor único en estado de carga

Figura 51

Indicador de valor único con error en su petición

4. Gráficos e indicadores de múltiples valores: Componentes visuales que permiten representar conjuntos de datos complejos, como series temporales o comparaciones. Se emplean gráficos de líneas, barras y áreas, acompañados de leyendas o etiquetas de valores. Estos componentes utilizan la paleta personalizada definida en la selección de colores y están optimizados para mantener una buena legibilidad en cualquier modo de visualización.

Figura 52

Gráfico de línea

Figura 53

Gráfico de línea con tooltip activo

Figura 54

Gráfico de líneas múltiples

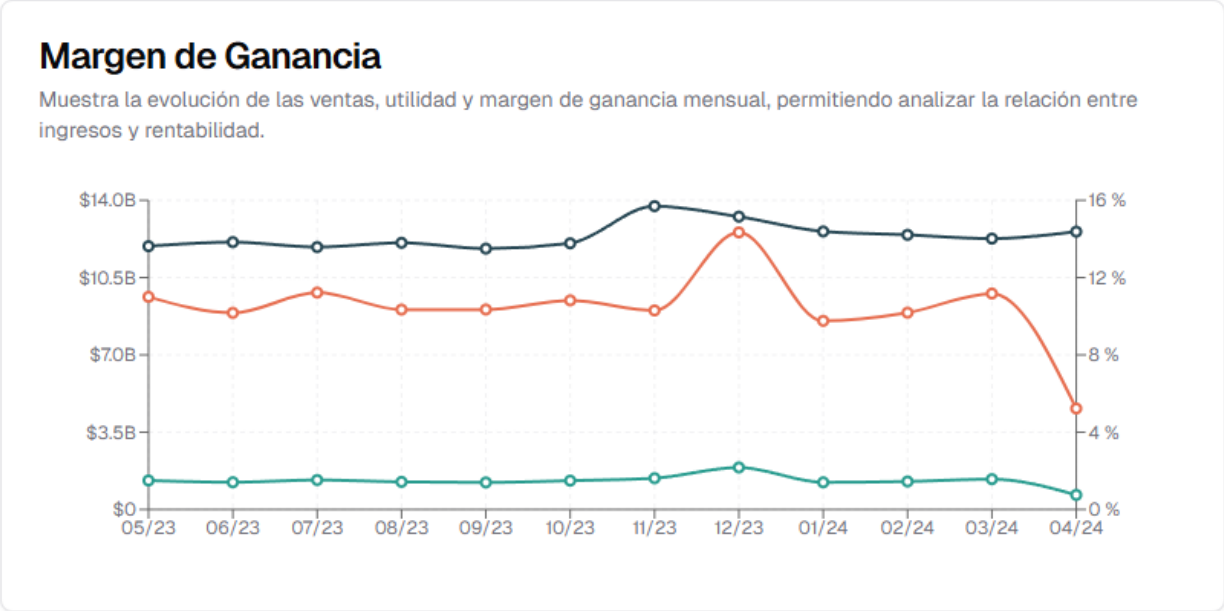


Figura 55

Gráfico de líneas múltiples con tooltip activo

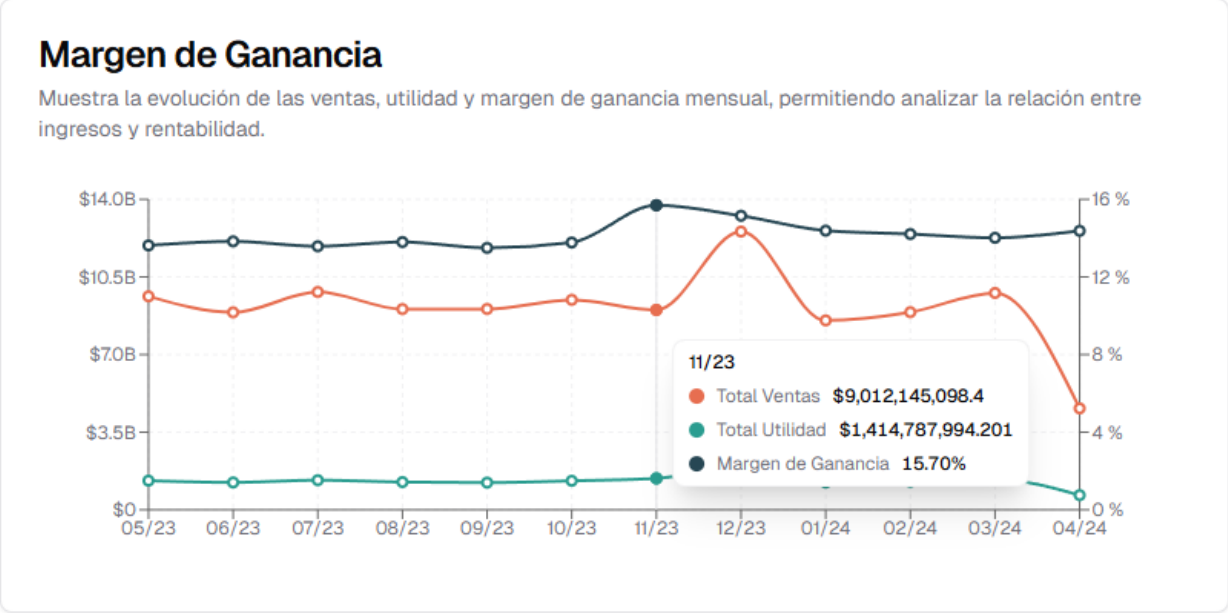


Figura 56

Gráfico de columnas

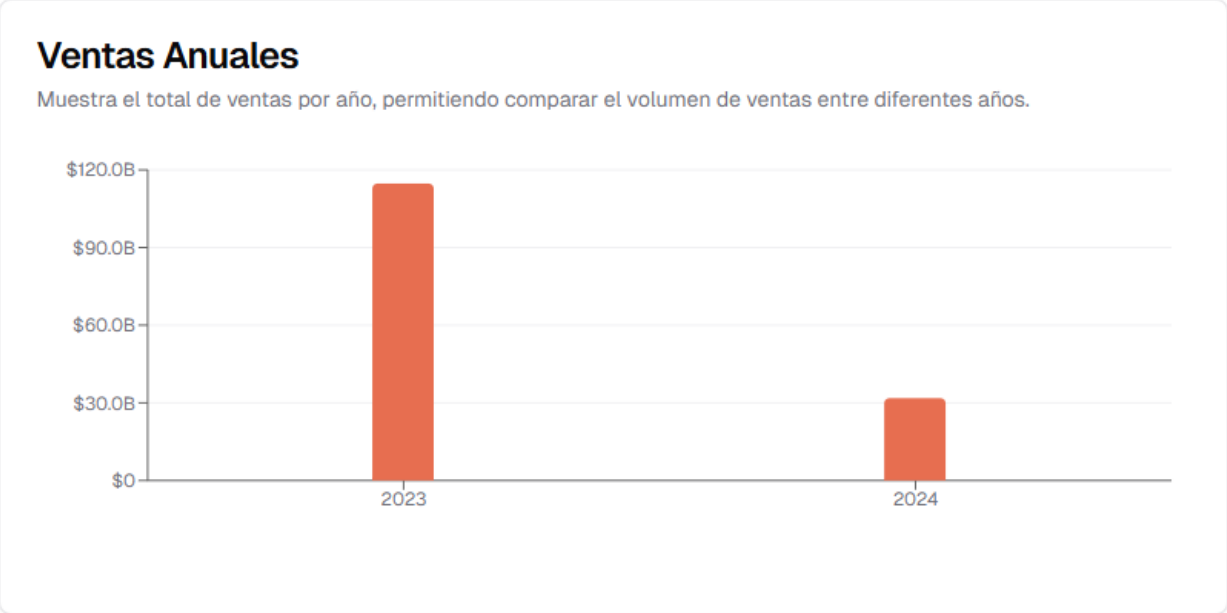


Figura 57

Gráfico de columnas con tooltip activo

Figura 58

Gráfico combinado

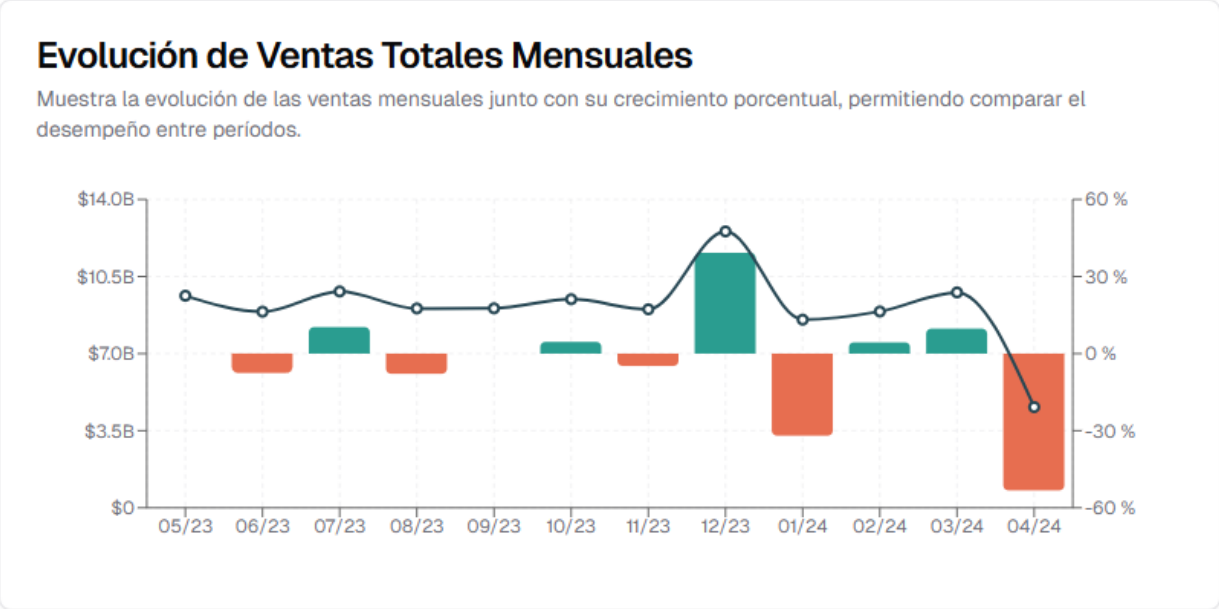


Figura 59

Gráfico combinado con tooltip activo

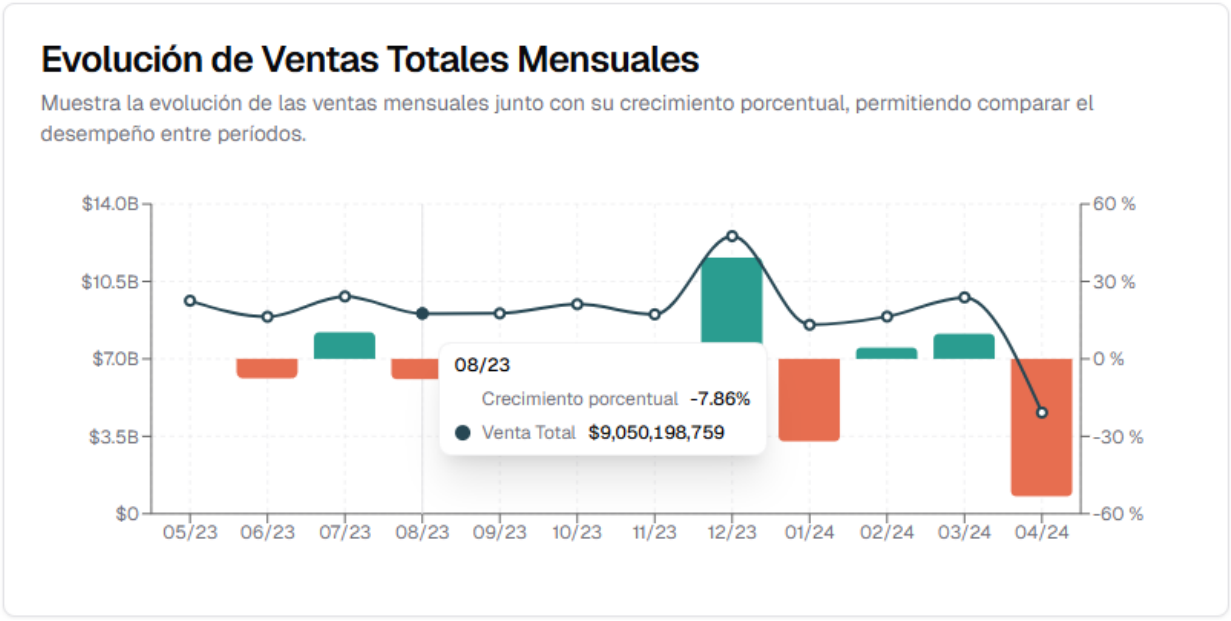
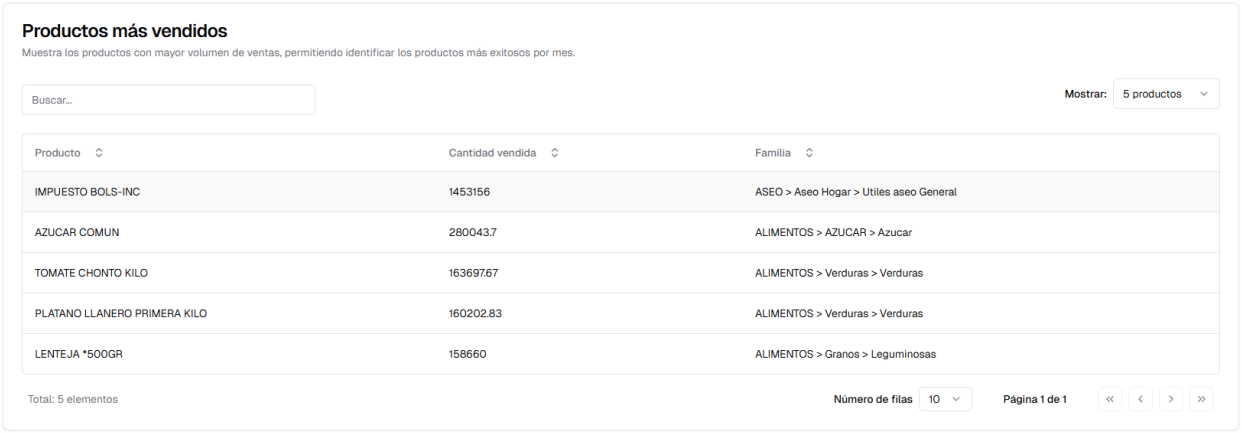


Figura 60

Tabla con lista de valores



5.5.1.7 Modo Oscuro

Las siguientes capturas muestran ejemplos del modo oscuro, evidenciando cómo la plataforma se adapta a diferentes configuraciones visuales.

Figura 61

Vista de indicadores de desempeño en modo oscuro

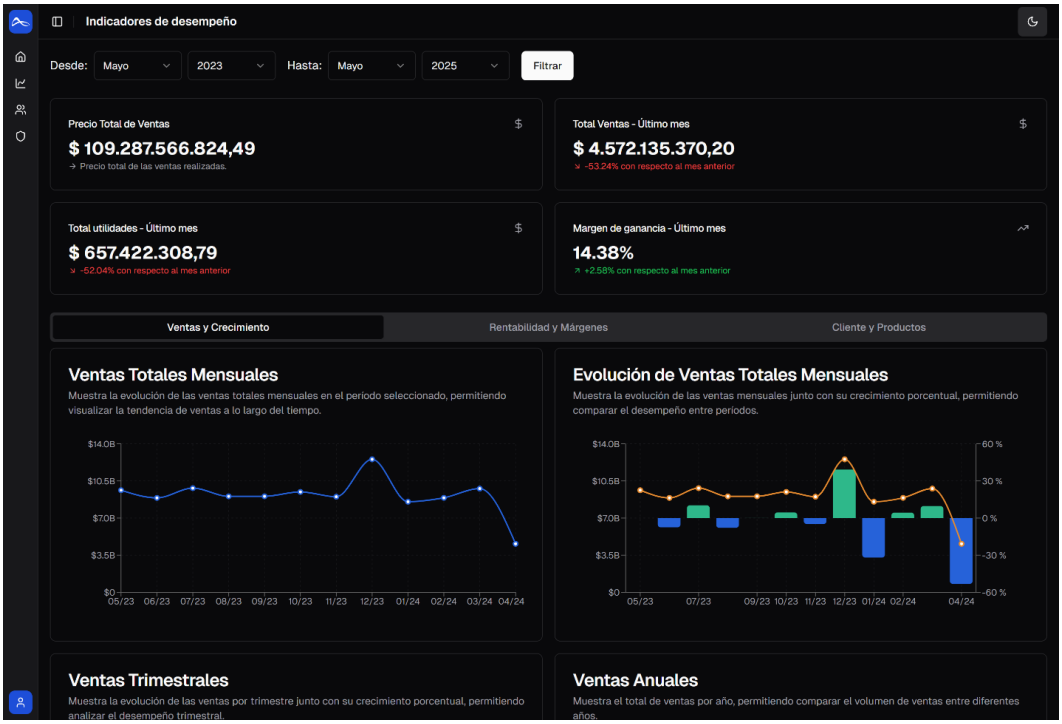


Figura 62

Vista de inicio en modo oscuro

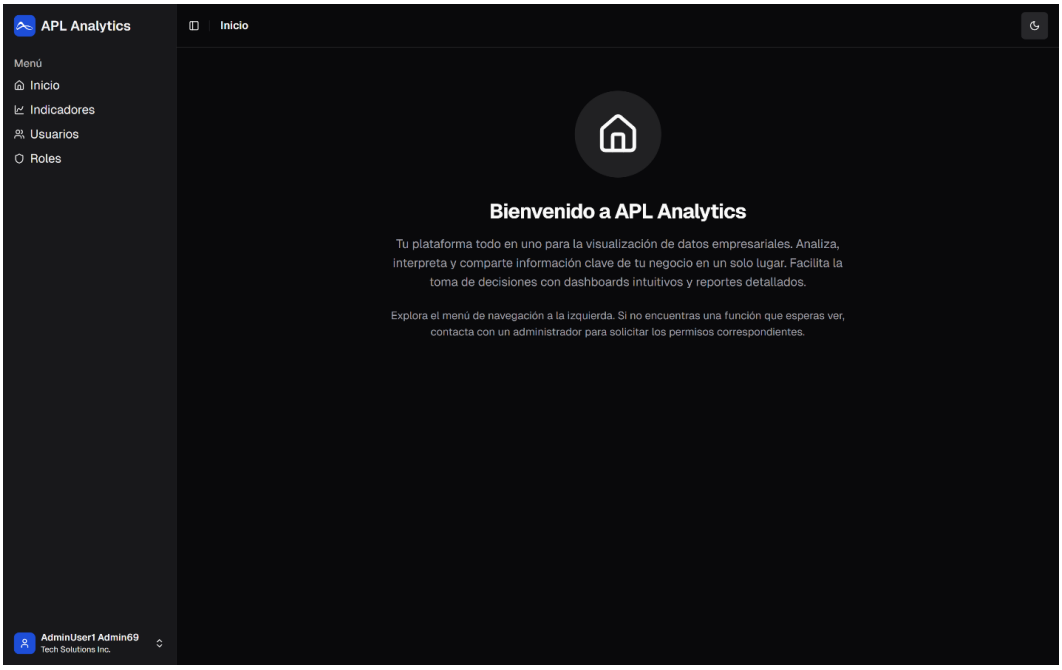


Figura 63

Vista de gestión de usuarios en modo oscuro

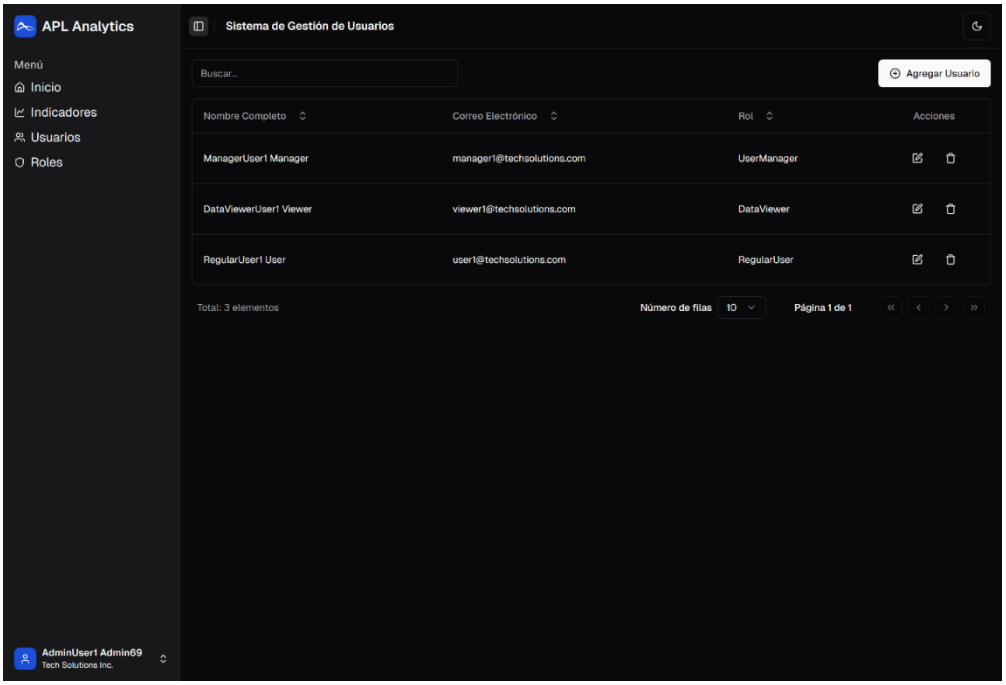
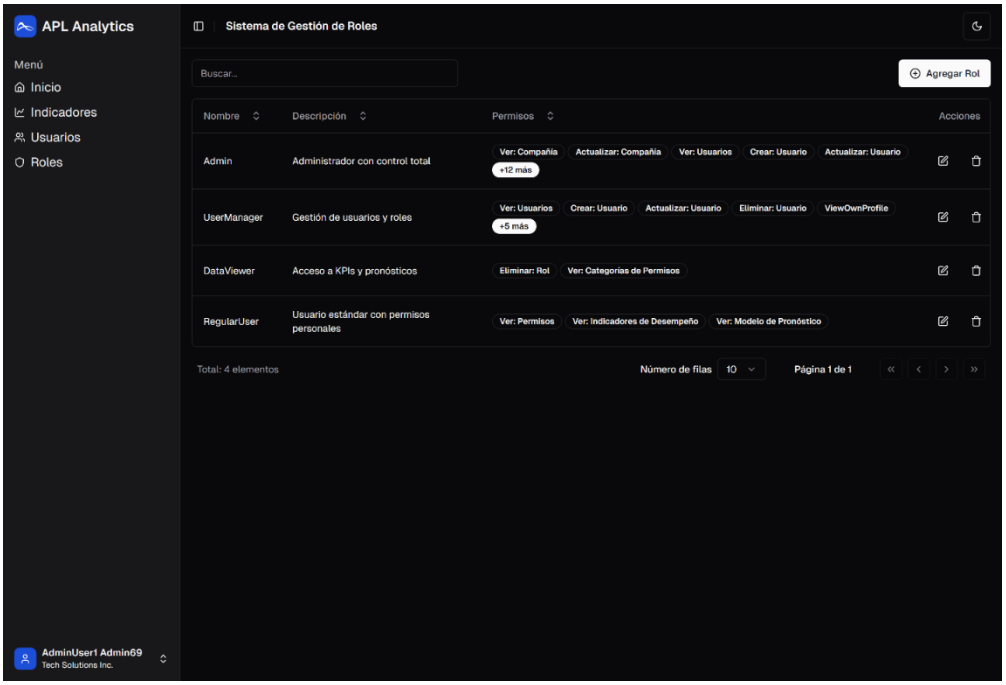


Figura 64

Vista de gestión de roles en modo oscuro



5.5.1.8 Adaptabilidad responsiva

Las siguientes capturas muestran ejemplos de la visualización de la plataforma en dispositivos móviles, evidenciando cómo la interfaz se adapta a diferentes tamaños de pantalla para mantener la usabilidad y consistencia visual.

Figura 65

Vista de inicio de sesión en dispositivos móviles

Figura 66

Menú lateral en dispositivos móviles

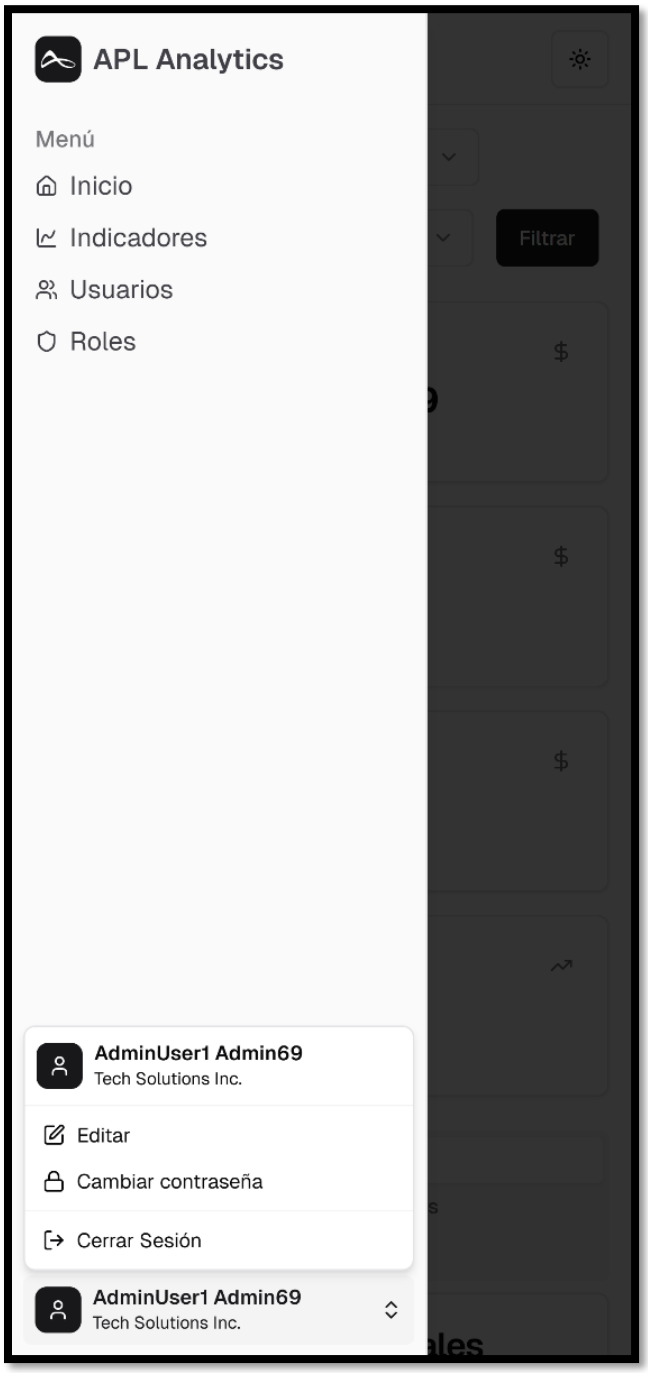
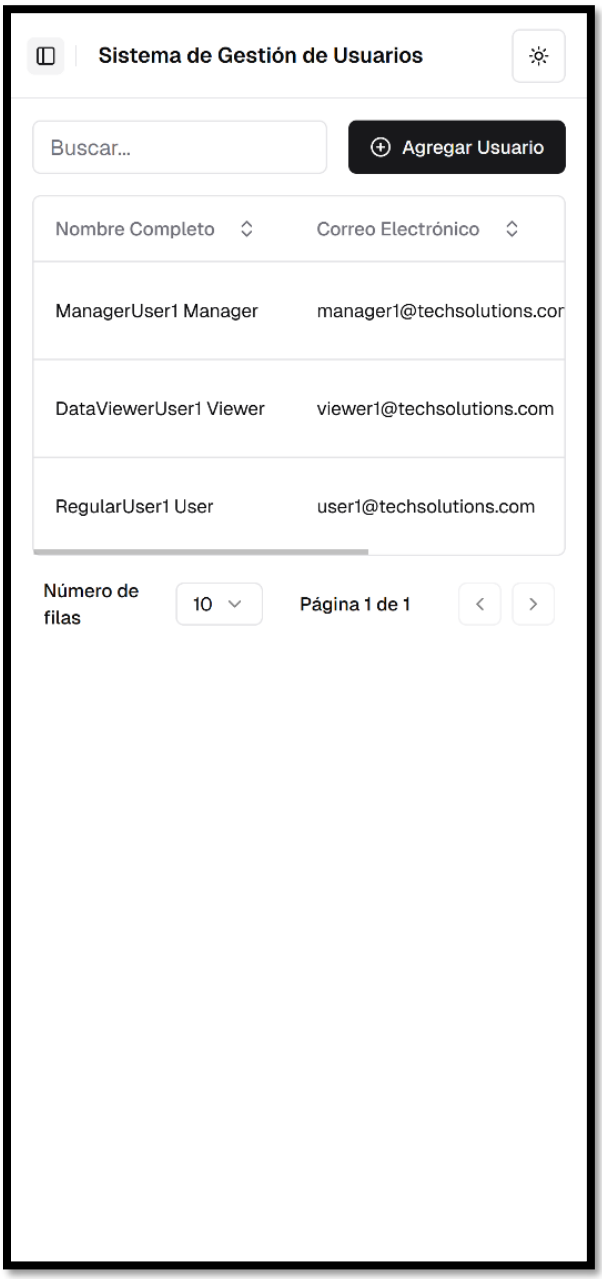


Figura 67

Vista de inicio en dispositivos móviles

Figura 68

Vista de gestión de usuarios en dispositivos móviles



5.5.2 Proceso ETL (Extracción, Transformación y Carga)

Como parte fundamental del desarrollo del proyecto, se llevó a cabo un proceso de integración de datos que permitió extraer, trasladar y adaptar la información desde la base de datos original hacia una nueva estructura optimizada. Este proceso consistió en tres etapas clave:

1. Se realizó la recolección de los datos necesarios desde la base principal.
2. Se aplicaron ajustes y adaptaciones a la información para asegurar su compatibilidad con la nueva estructura.
3. Se procedió a cargar los datos en la base de datos destino.

En cuanto a la etapa de recolección, se extrajo la información directamente desde la base de datos principal, lo que permitió trabajar con datos consistentes, actualizados y completos.

Durante la fase de adaptación, se realizaron transformaciones específicas como normalización de formatos, limpieza de registros y adecuación de campos para que cumplieran con el nuevo esquema relacional. Este paso fue esencial para asegurar la integridad y coherencia de la información una vez cargada.

Finalmente, durante el proceso de carga, algunas tablas pudieron transferirse directamente debido a su tamaño reducido y estructura sencilla. Sin embargo, en el caso de tablas con un volumen considerable de registros —como por ejemplo *SalidasDeMercancia* y *SalidasDeMercanciaDetalle*, que superaban los 22 millones de filas— fue necesario realizar una carga progresiva. Para ello, se optó por un enfoque segmentado, cargando los datos de manera trimestral por años, lo que permitió optimizar los tiempos de procesamiento, reducir errores y mantener la estabilidad del software durante la migración. Este enfoque estructurado permitió mantener la eficiencia y confiabilidad en el proceso de integración, asegurando que los datos en la nueva base estén disponibles y listos para su análisis y uso operativo.

Figura 69

Conexiones entre Bases de Datos

```python
import pandas as pd
from sqlalchemy import create_engine

# Configuración de conexión
DB_SERVER_ORIGEN = "SISTEMAS04"
DB_DATABASE_ORIGEN = "APLBDS_MTRO"
DB_USERNAME_ORIGEN = "apladmin"
DB_PASSWORD_ORIGEN = "1234"
connection_string_origen = f"mssql+pyodbc://{DB_USERNAME_ORIGEN}:{DB_PASSWORD_ORIGEN}@{DB_SERVER_ORIGEN}/{DB_DATABASE_ORIGEN}?driver=ODBC+Driver+17+for+SQL
engine_origen = create_engine(connection_string_origen)

DB_SERVER_DESTINO = "APLSERVIDOR"
DB_DATABASE_DESTINO = "Prueba"
DB_USERNAME_DESTINO = "apladmin"
DB_PASSWORD_DESTINO = "1234"
connection_string_destino = f"mssql+pyodbc://{DB_USERNAME_DESTINO}:{DB_PASSWORD_DESTINO}@{DB_SERVER_DESTINO}/{DB_DATABASE_DESTINO}?driver=ODBC+Driver+17+fo
engine_destino = create_engine(connection_string_destino)
```

Para realizar el traslado de la información desde la base de datos original hacia la nueva, se configuraron conexiones directas a ambas fuentes mediante scripts en Python, utilizando la librería sqlalchemy. Esto permitió establecer un canal de comunicación confiable entre el sistema origen y el sistema destino. En este caso, la base de datos principal llamada APLBDS_MTRO, alojada en el servidor sistemas04 contenía los datos históricos del sistema.

Por otro lado, la nueva base de datos llamada ProyectoFenix, en el servidor aplservidor fue el destino donde se cargó toda la información reorganizada. Las credenciales de acceso fueron gestionadas de forma segura en el script, y se usó el controlador ODBC correspondiente (ODBC Driver 17 for SQL Server) para asegurar la compatibilidad. Una vez establecidas las conexiones con ambas bases, se prepararon motores de lectura y escritura (eng_src y eng_dst) que permitieron operar sobre las tablas utilizando consultas SQL directamente desde Python.

Figura 70

Proceso de Transformación y Carga.

```
CARGA_DATOS_TABLA_SALIDA_MERCANCIA_DETALLE

import pandas as pd
from sqlalchemy import create_engine
from datetime import datetime

def get_engines():
    eng_src = create_engine(
        "mssql+pyodbc://apladmin:1234@SISTEMAS04/APLBDS_MTRO"
        "?driver=ODBC+Driver+17+for+SQL+Server"
    )
    eng_dst = create_engine(
        "mssql+pyodbc://apladmin:1234@APLSERVIDOR/Prueba"
        "?driver=ODBC+Driver+17+for+SQL+Server",
        fast_executemany=True
    )
    return eng_src, eng_dst


def consulta(f_ini: datetime, f_fin: datetime) -> str:
    return f"""
        SELECT
            smd.ProductoId,
            sm.SalidaId  AS SalidasDeMercancia_Id,
            smd.CantidadSalida,
            sm.EmpresaId
        FROM dbo.SalidasDeMercanciaDetalle AS smd
        INNER JOIN dbo.SalidasDeMercancia   AS sm
                ON sm.NoDocumento = smd.NoDocumento
        WHERE sm.FechaDeSalida >= '{f_ini:%Y-%m-%d}'
          AND sm.FechaDeSalida <  '{f_fin:%Y-%m-%d}'
          AND sm.TipoDocumento IS NOT NULL;      -- <--- sincronizado con maestro
    """


def cargar_anio(anio: int, eng_src, eng_dst) -> None:
    cortes = [
        datetime(anio, 1, 1),
        datetime(anio, 4, 1),
        datetime(anio, 7, 1),
        datetime(anio, 10, 1),
        datetime(anio + 1, 1, 1),
    ]

    total = 0
    for i in range(4):
        f_ini, f_fin = cortes[i], cortes[i + 1]
        etiqueta = f"{anio}-Q{i + 1}"

        df = pd.read_sql(consulta(f_ini, f_fin), eng_src)

        df.to_sql(
            "SalidasDeMercanciaDetalle",
            eng_dst,
            if_exists="append",
            index=False
        )

        filas = len(df)
        print(f"{etiqueta:<8} → {filas:,} filas")
        total += filas

    print(f"\n✅ TOTAL {anio}: {total:,} filas transferidas.\n")


if __name__ == "__main__":
    eng_src, eng_dst = get_engines()
    cargar_anio(2022, eng_src, eng_dst)
```

Para manejar el volumen masivo de datos, especialmente en tablas como *SalidasDeMercancia* y *SalidasDeMercanciaDetalle* —las cuales superaban los 22 millones de registros— se optó por un enfoque de carga progresiva basado en trimestres.Este procedimiento consistió en definir rangos de fechas trimestrales para un año determinado. A partir de esto, se ejecutaba una consulta SQL que extraía los registros correspondientes al primer trimestre, luego al segundo, y así sucesivamente. En cada ciclo, los datos se cargaban en la nueva base de datos utilizando la función `to_sql` de la librería `pandas`, que permite insertar grandes cantidades de

datos de forma eficiente. Antes de la carga, se contempló la posibilidad de realizar transformaciones adicionales sobre los datos, como ajustes de valores nulos o normalización de campos. Aunque en algunos casos no fueron necesarias, se dejó el bloque preparado para futuras modificaciones, manteniendo el código modular y escalable. Cada ejecución trimestral reportaba la cantidad de filas procesadas, permitiendo así tener control y trazabilidad sobre el avance del proceso. Finalmente, al completar los cuatro trimestres de un año, se mostraba un resumen total de registros transferidos, confirmando el éxito de la operación.

5.5.3 Backend

A continuación, se describe la implementación del backend principal del sofware, el cual se encargó de gestionar la lógica de negocio, el acceso a datos y la exposición de funcionalidades a través de una API. Este backend fue desarrollado siguiendo una arquitectura por capas que facilita la escalabilidad y el mantenimiento del software.

5.5.3.1 Backend de Gestión y Control

El desarrollo del backend del software se llevó a cabo utilizando .NET, siguiendo una arquitectura por capas bajo los principios de arquitectura limpia. El objetivo de este módulo es gestionar la configuración base del software, incluyendo usuarios, roles, permisos y empresas, así como administrar la autenticación, autorización y control de acceso.

Este backend actúa como puente principal de comunicación entre el cliente (frontend) y los distintos componentes del software, incluyendo un segundo backend implementado en FastAPI para el procesamiento de indicadores y modelos predictivos. Para esto, se utilizó HttpClient dentro del servicio principal, permitiendo realizar solicitudes HTTP internas a dicho backend, mientras se mantiene la centralización del control de autenticación y permisos.

1. Estructura de la solución en .NET

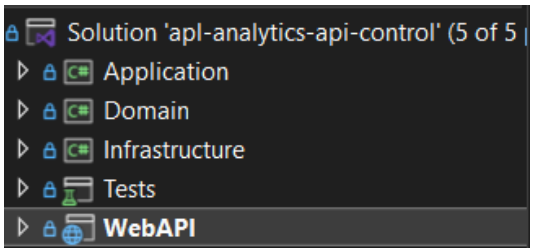
El proyecto fue estructurado como una solución de .NET compuesta por cuatro proyectos independientes, cada uno representando una capa lógica de la aplicación:

- WebApi (presentación)
- Application (servicio o lógica de negocio)
- Infrastructure (acceso a datos)
- Domain (entidades del negocio)

Esta separación permite un desarrollo escalable, organizado y desacoplado, cumpliendo con los principios de responsabilidad única, separación de intereses y facilidad para pruebas automatizadas.

Figura 71

Estructura del proyecto en Visual Studio



2. Organización de dependencias entre capas

La organización de dependencias se realizó de acuerdo con los principios de arquitectura limpia:

- Domain es la capa más interna y no depende de ninguna otra. Contiene las entidades del dominio y las interfaces de los repositorios.
- Infrastructure depende de Domain, ya que implementa las interfaces de acceso a datos definidas en el dominio y necesita conocer las entidades.

- Application depende tanto de Infrastructure como de Domain, ya que implementa la lógica de negocio utilizando los repositorios implementados y las entidades.
- WebApi depende únicamente de Application, de forma que no accede directamente a la base de datos ni conoce detalles de la implementación interna.

Esta configuración de dependencias permite mantener una estructura limpia, desacoplada y mantenible, facilitando los cambios o reemplazos de tecnologías en las capas externas sin afectar la lógica central del software.

3. Uso de bibliotecas y herramientas por capa

Cada capa del proyecto utiliza únicamente las bibliotecas necesarias para su responsabilidad:

- Domain: No utiliza bibliotecas externas, asegurando su independencia.
- Infrastructure: Se integró Dapper, una biblioteca ligera de mapeo objeto-relacional, para ejecutar consultas SQL directas de manera eficiente.
- Aplicación: Se utilizó BCrypt.Net para el hashing de contraseñas, contribuyendo a la protección de las credenciales. Se configuró HttpClient para permitir la comunicación con el backend secundario de FastAPI, encargado del análisis de datos y predicciones.
- WebApi: Se configuró Swagger para la documentación y prueba de los endpoints. Se implementaron middlewares personalizados, además del uso de los middlewares integrados UseAuthentication y UseAuthorization.

4. Middlewares para autenticación y autorización

En la capa de WebApi se desarrolló un middleware personalizado para controlar el acceso a la aplicación mediante JWT Bearer Tokens. Este middleware extrae y valida los claims del token,

consulta los servicios del backend para verificar el usuario y sus permisos, y redirige la solicitud al endpoint correspondiente solo si el acceso es válido.

Además, se utilizaron los middlewares estándar de .NET:

- UseAuthentication: Encargado de autenticar al usuario mediante el token JWT.
- UseAuthorization: Valida que el usuario tenga los permisos necesarios para ejecutar una acción determinada.

Este software permite asegurar cada petición a la API y verificar que solo usuarios autorizados accedan a los recursos definidos.

5. Endpoints

El backend del módulo de Gestión y Control expone una serie de endpoints RESTful que permiten realizar operaciones de autenticación, gestión de usuarios, roles, permisos, empresas, entre otras funcionalidades. Estos endpoints siguen el estándar HTTP y están organizados por recursos, utilizando los verbos adecuados para cada operación (GET, POST, PUT, DELETE).

Cada endpoint está protegido mediante JWT Bearer Authentication, y la autorización se gestiona según los permisos definidos para cada rol dentro del software. A continuación, se presenta una tabla con los endpoints más relevantes implementados:

Tabla 26

Endpoints del controlador de usuarios

Método	Ruta	Descripción	Respuesta esperada
GET	/byId/{userId}	Obtener usuario por ID	Objeto `UserDto` con los datos del usuario solicitado
GET	/api/[controller]	Obtener los usuarios	Lista de objetos `UserDto` de la empresa del usuario autenticado
POST	/api/[controller]	Crear un nuevo usuario	Objeto `UserDto` del usuario creado
PUT	/{userId}	Actualizar usuario por ID	Objeto `UserDto` actualizado
PATCH	/{userId}/deactivate	Desactivar usuario	Mensaje indicando que el usuario se ha desactivado

GET	/me	Obtener perfil del usuario autenticado	Objeto `UserDto` del propio usuario
PUT	/me	Actualizar perfil propio	Objeto `UserDto` actualizado con los nuevos datos personales
PUT	/change-my-password	Cambiar la contraseña propia	Código HTTP 204 sin contenido (indica éxito sin respuesta)

Nota. UserDto es un objeto de transferencia de datos (DTO) que representa parte de los campos de la entidad Usuarios de la base de datos, excluyendo información sensible como contraseñas. Se utiliza para exponer datos del usuario de forma segura en las respuestas de la API.

Tabla 27

Endpoint del controlador de roles

Método	Ruta	Descripción	Respuesta esperada
GET	/api/role/{roleId}	Obtener rol por ID	Objeto RoleDto del rol solicitado
GET	/api/role	Obtener los roles	Lista de objetos RoleDto de la empresa del usuario autenticado
POST	/api/role	Crear un nuevo rol	Objeto RoleDto del rol creado.
PUT	/api/role/{roleId}	Actualizar rol por ID	Objeto anónimo con éxito, mensaje y rol actualizado.
PUT	/api/role/disable/{roleId}	Desactivar un rol	Objeto con Success=true y mensaje si se desactiva correctamente

Nota. El objeto RoleDto representa los datos públicos relevantes de la entidad Role en la base de datos.

Los siguientes endpoints no requieren autenticación mediante tokens JWT. Esto se debe a que permiten acciones básicas como solicitar restablecimientos de contraseña o iniciar sesión, funciones que deben estar disponibles para usuarios no autenticados.

Tabla 28

Endpoints públicos

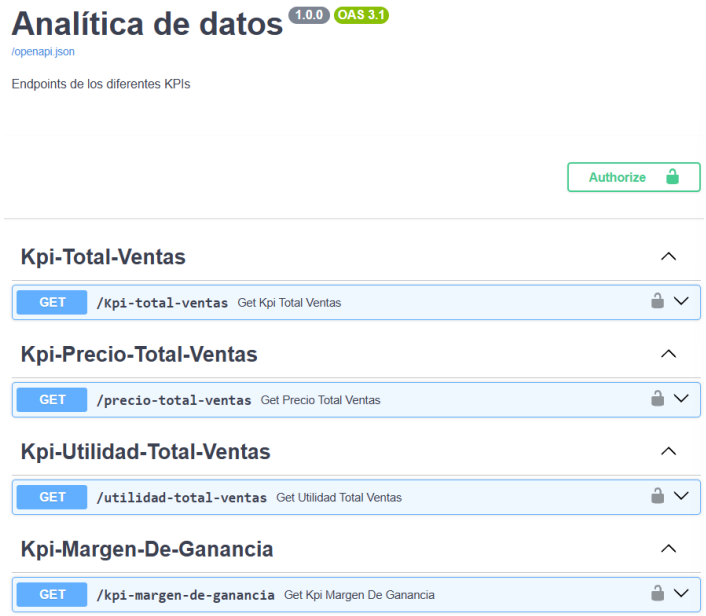
Método	Ruta	Descripción	Respuesta esperada
POST	/login	Iniciar sesión	Token de acceso y datos del usuario autenticado
POST	/request-password-reset	Solicitar restablecimiento de contraseña	Código HTTP 204 sin contenido
POST	/confirm-password-reset	Confirmar restablecimiento de contraseña	Código HTTP 204 sin contenido

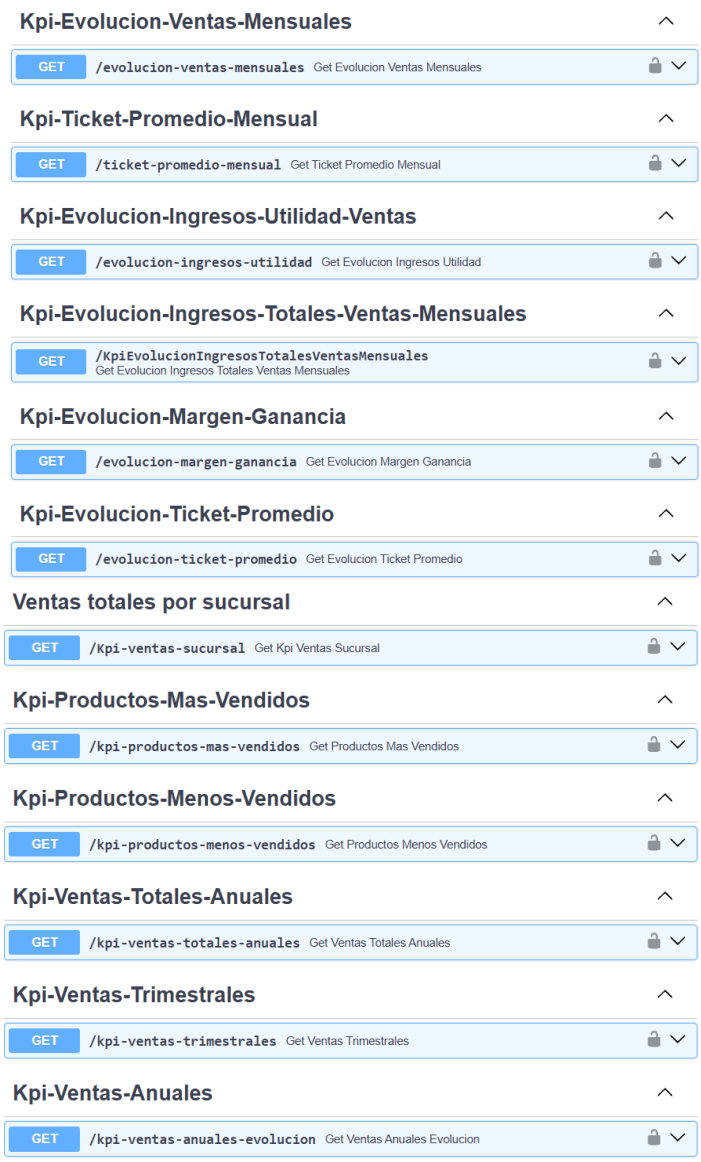
5.5.3.2 Indicadores clave de desempeño (KPI)

En esta sección se explorarán los indicadores clave de rendimiento (KPIs) aplicados al área de ventas, fundamentales para monitorear y optimizar el desempeño comercial de la organización. Estos KPIs permiten evaluar desde la efectividad de las ventas hasta la rentabilidad de productos, proporcionando datos concretos para tomar decisiones estratégicas.

Figura 72

Endpoints KPIS





Nota. En las imágenes anteriores podemos observar los endpoints de los KPI

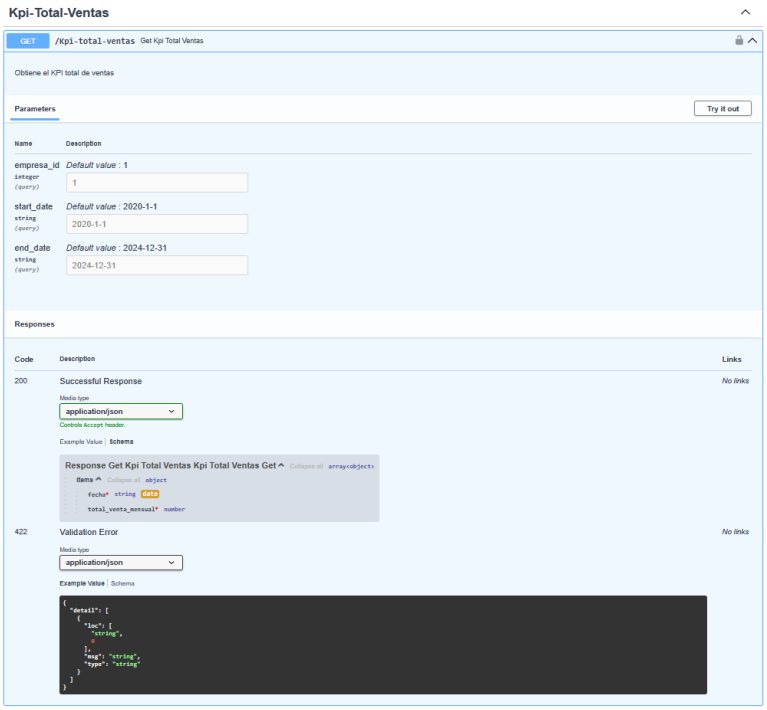
1. KPIS de Rendimiento General de Ventas

El indicador referenciado en la figura 73 refleja el rendimiento general de las ventas al mostrar el volumen total de ingresos generados en un periodo determinado. Se calcula mediante la suma acumulada del valor monetario de todas las transacciones realizadas, lo que permite tener

una visión precisa del comportamiento comercial de la organización. Es una métrica clave para evaluar el crecimiento y la eficiencia del área de ventas.

Figura 73

KPI Total Ventas

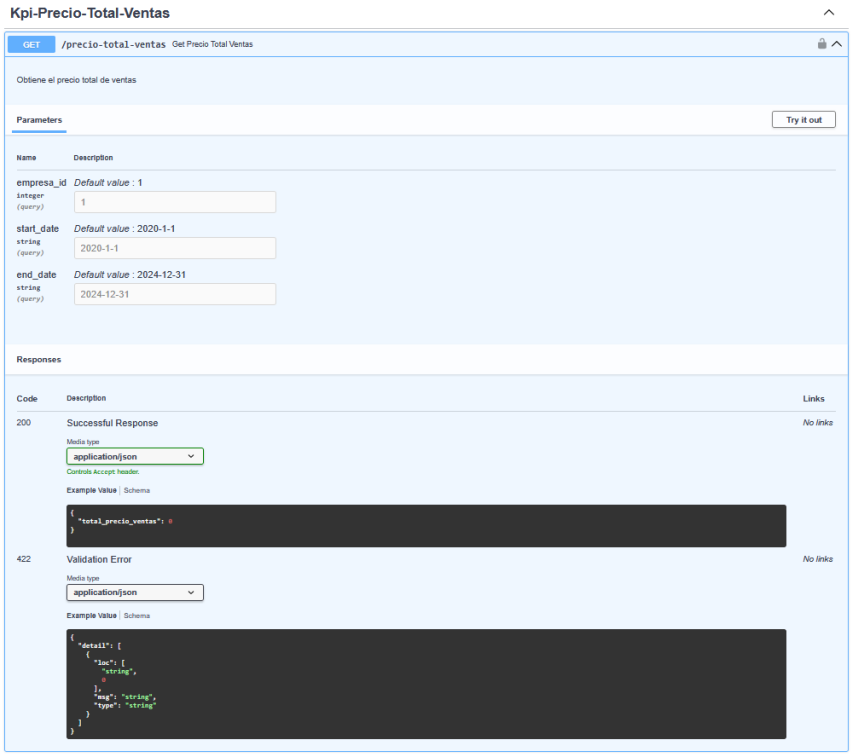


El indicador referenciado en la figura 74 muestra el valor económico total generado por la venta de productos, considerando el precio individual de cada ítem vendido. A diferencia de un promedio, este KPI representa la suma completa de los precios de los productos comercializados en un periodo determinado, sin tener en cuenta la cantidad de unidades por tipo de producto. Es

útil para obtener una visión detallada del ingreso bruto derivado de las ventas y permite identificar el impacto total del portafolio ofrecido en términos de valor.

Figura 74

KPI Precio Total Ventas

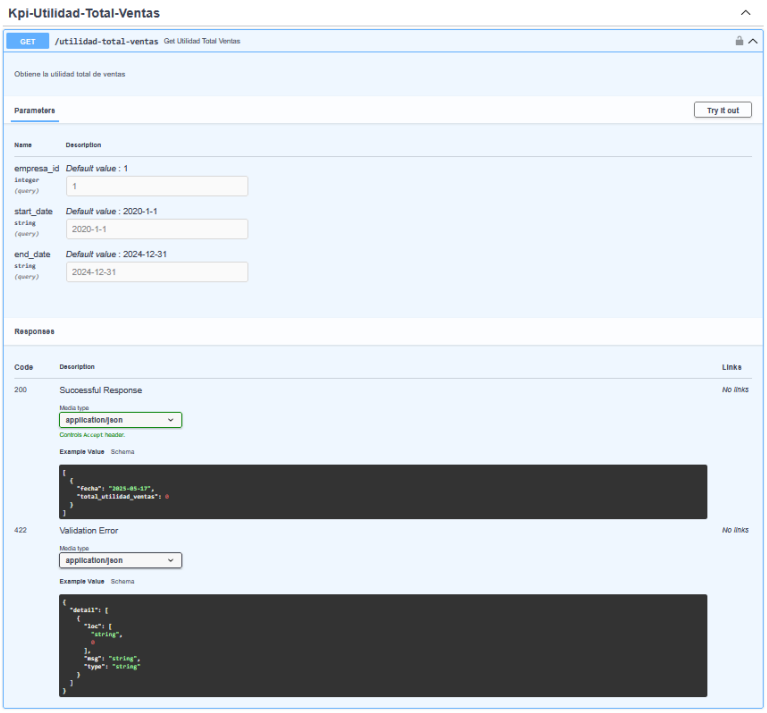


El indicador referenciado en la figura 75 mide la ganancia económica obtenida a partir de las ventas, es decir, la diferencia entre los ingresos generados por la venta de productos y los costos asociados a dichos productos. Se calcula restando el costo de los bienes vendidos (COGS) al total de ventas realizadas en un periodo determinado. Este KPI permite evaluar la rentabilidad operativa del negocio y es clave para identificar si las estrategias comerciales están generando márgenes

positivos. Su análisis frecuente facilita la toma de decisiones relacionadas con ajustes de precios, control de costos y optimización del portafolio de productos.

Figura 75

KPI Utilidad Total Ventas



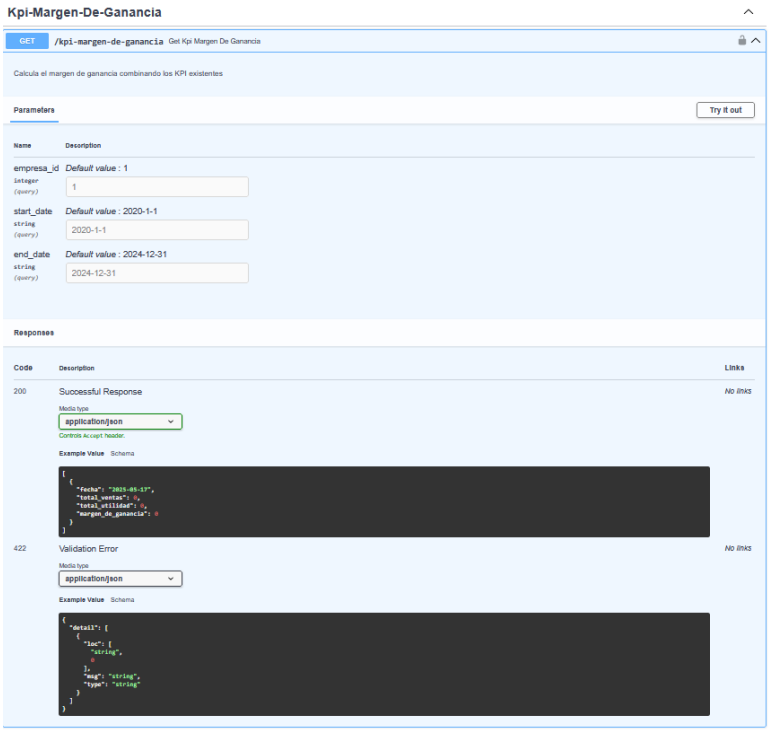
1. KPIS de Rentabilidad.

El indicador referenciado en la figura 76 muestra el porcentaje de ganancia que obtiene la empresa por cada unidad monetaria de venta. Se calcula dividiendo la utilidad total de ventas entre el total de ingresos por ventas y multiplicando el resultado por 100. A diferencia de la utilidad total, que ofrece un valor absoluto, el margen de ganancia proporciona una visión relativa de la rentabilidad, permitiendo comparar el desempeño entre diferentes productos, periodos o estrategias comerciales. Un margen alto indica que la empresa está generando una buena ganancia en relación con sus costos, mientras que un margen bajo puede reflejar precios poco competitivos,

costos elevados o problemas en la estrategia comercial. Es un KPI clave para evaluar la eficiencia financiera del negocio.

Figura 76

KPI Margen de Ganancia

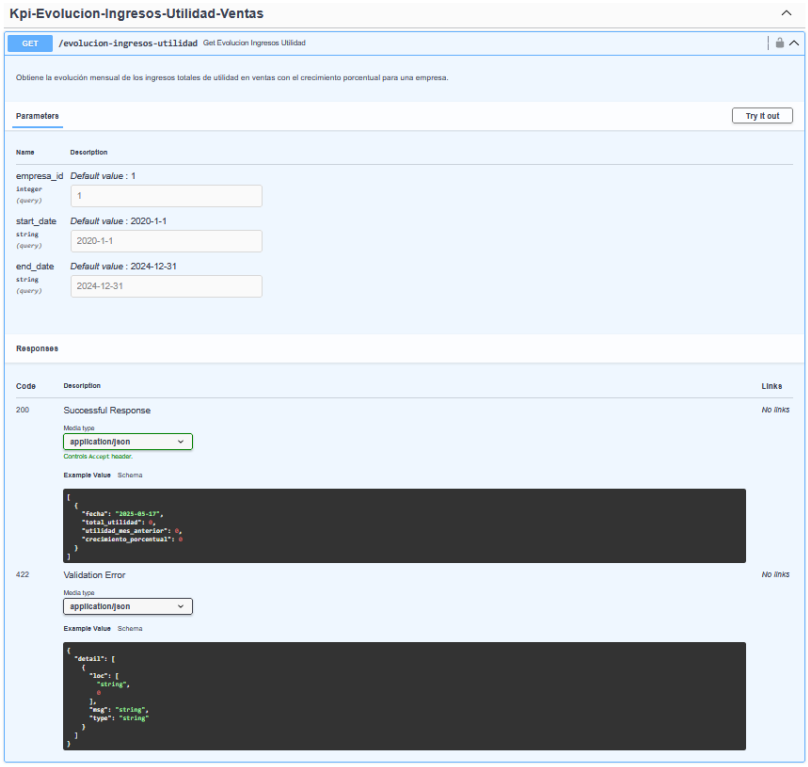


El indicador referenciado en la figura 77 permite analizar la tendencia a lo largo del tiempo tanto de los ingresos generados por ventas como de la utilidad obtenida. Su objetivo es identificar comportamientos positivos o negativos en el desempeño financiero del negocio, comparando los resultados de distintos periodos (diarios, mensuales, trimestrales, etc.). Este KPI es especialmente útil para detectar patrones estacionales, evaluar el impacto de campañas comerciales o cambios de estrategia, y anticipar posibles riesgos o oportunidades. Al combinar ingresos y utilidad, ofrece

una visión integral sobre el crecimiento sostenido del negocio y su rentabilidad a lo largo del tiempo.

Figura 77

KPI Evolución Ingresos Utilidad Ventas



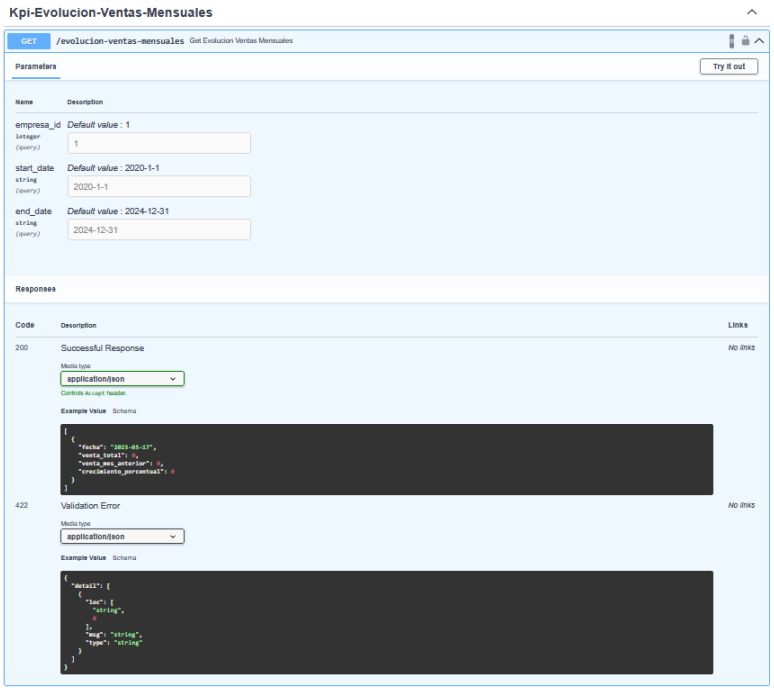
2. KPIS de Comportamiento de Ventas

El indicador referenciado en la figura 78 permite hacer un seguimiento del comportamiento de las ventas a lo largo del tiempo, específicamente en un análisis mensual. Su propósito es identificar tendencias, estacionalidades y variaciones significativas en el volumen de ventas, lo que facilita la toma de decisiones estratégicas. Al observar la evolución mes a mes, es posible detectar aumentos sostenidos, caídas inesperadas o el impacto de acciones comerciales específicas.

Este KPI es clave para evaluar el crecimiento del negocio, planificar presupuestos y ajustar estrategias de marketing, ventas y producción con base en datos reales y actualizados.

Figura 78

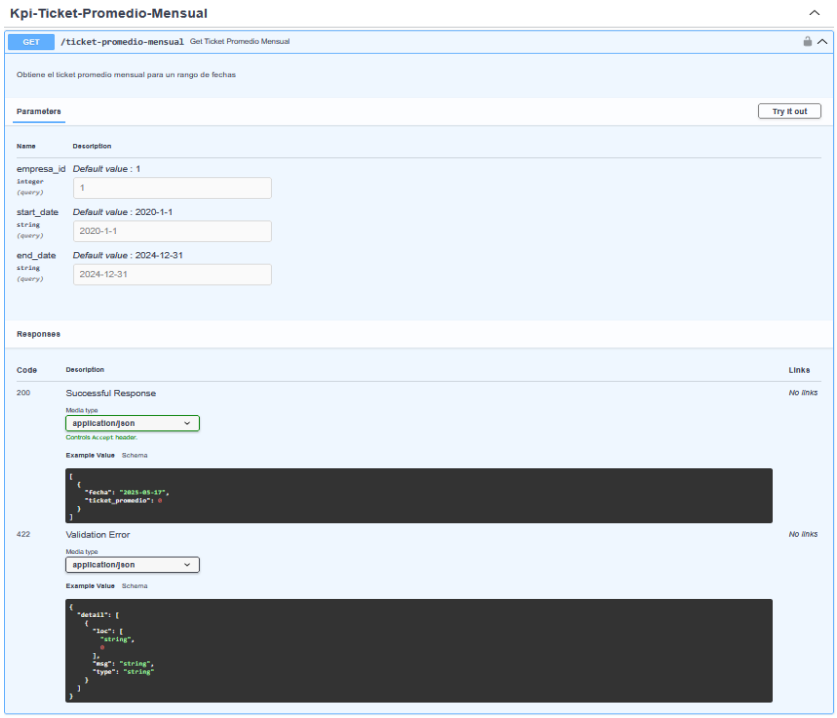
KPI Evolución Ventas Mensuales



El indicador referenciado en la figura 79 muestra el valor promedio de cada transacción realizada durante un mes, es decir, cuánto gasta en promedio un cliente por compra. Se calcula dividiendo el total de ingresos por ventas en un mes entre la cantidad total de transacciones efectuadas en ese mismo periodo. El KPI de ticket promedio mensual es útil para entender el comportamiento de consumo de los clientes y evaluar el impacto de estrategias como promociones, ventas cruzadas o aumentos de precios. Su seguimiento permite identificar oportunidades para incrementar el valor de cada venta y mejorar la rentabilidad del negocio.

Figura 79

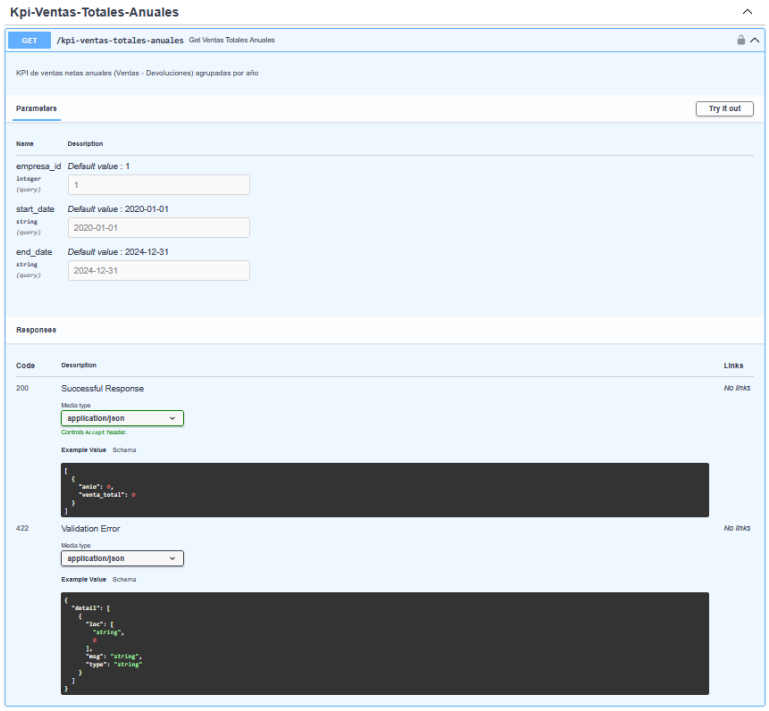
KPI Ticket Promedio Mensual



El indicador referenciado en la figura 80 refleja el volumen total de ventas generadas por la empresa durante un año completo, expresado en términos monetarios. Permite evaluar el rendimiento comercial desde una perspectiva global y de largo plazo, sirviendo como referencia clave para analizar el crecimiento del negocio, definir metas futuras y tomar decisiones estratégicas a nivel gerencial. Al consolidar los ingresos de cada mes del año, este KPI facilita comparaciones interanuales, la detección de tendencias generales y la medición del impacto de las estrategias comerciales implementadas durante el periodo.

Figura 80

KPI Ventas Totales Anuales

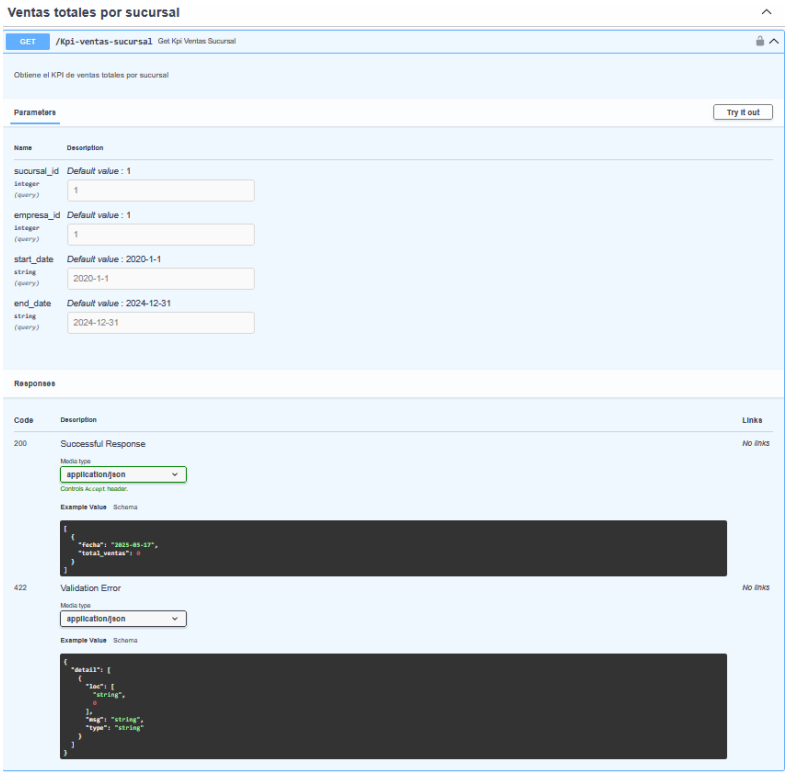


El indicador referenciado en la figura 81 muestra el volumen de ventas generadas individualmente por cada sucursal de la empresa durante un periodo determinado. Permite comparar el desempeño comercial entre distintas ubicaciones físicas, identificar cuáles están contribuyendo más al ingreso total y detectar oportunidades de mejora en aquellas con bajo rendimiento. El análisis de ventas por sucursal es fundamental para la toma de decisiones operativas y estratégicas, ya que facilita la asignación eficiente de recursos, el ajuste de inventarios

y la planificación de campañas comerciales específicas según el comportamiento de cada punto de venta.

Figura 81

Ventas por Sucursal

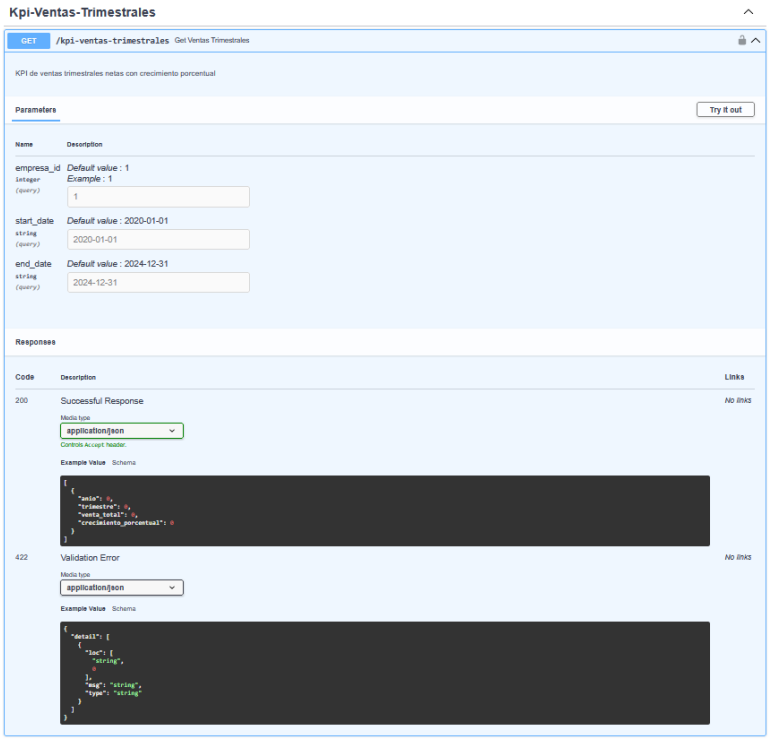


El indicador referenciado en la figura 82 permite evaluar el desempeño comercial de la empresa en periodos de tres meses, ofreciendo una visión intermedia entre el análisis mensual y anual. Al agrupar las ventas por trimestres, facilita la identificación de tendencias estacionales, el impacto de campañas o eventos específicos, y el seguimiento de objetivos financieros de corto y mediano plazo. El KPI de ventas trimestrales es útil para comparar el rendimiento entre diferentes

trimestres del mismo año o frente a años anteriores, proporcionando información clave para la planificación estratégica, la gestión presupuestaria y la toma de decisiones informadas.

Figura 82

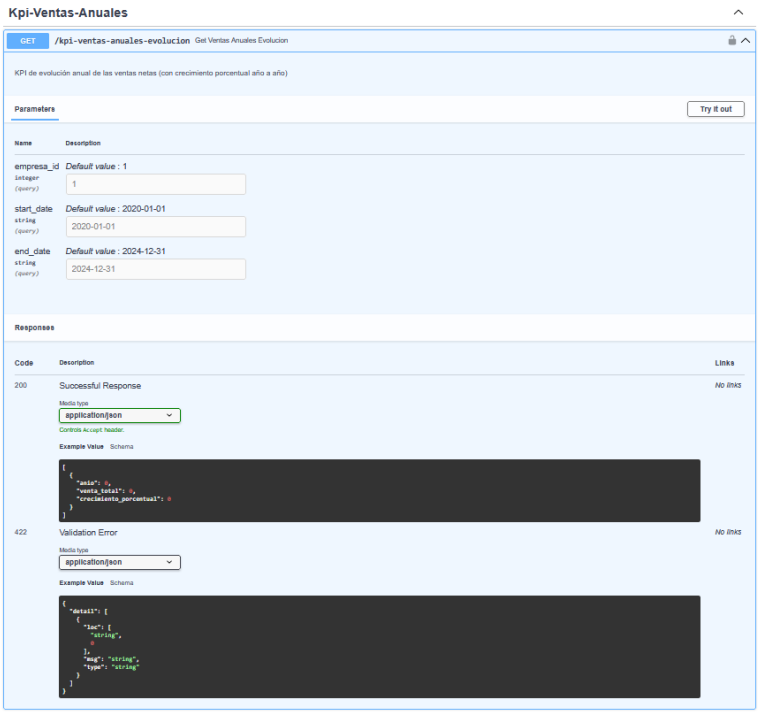
KPI de Ventas Trimestrales



El indicador referenciado en la figura 83 analiza la variación del volumen de ventas a lo largo de varios años, permitiendo observar la evolución del desempeño comercial en el tiempo. Su objetivo es identificar tendencias de crecimiento, estancamiento o disminución en los ingresos anuales, facilitando así la evaluación del impacto de estrategias comerciales, cambios en el mercado o factores externos relevantes. El KPI de evolución de ventas anuales es fundamental para la planificación estratégica a largo plazo, ya que proporciona una base sólida para establecer metas, estimar proyecciones futuras y tomar decisiones enfocadas en la sostenibilidad del negocio.

Figura 83

KPI de Evolución de Ventas Anuales

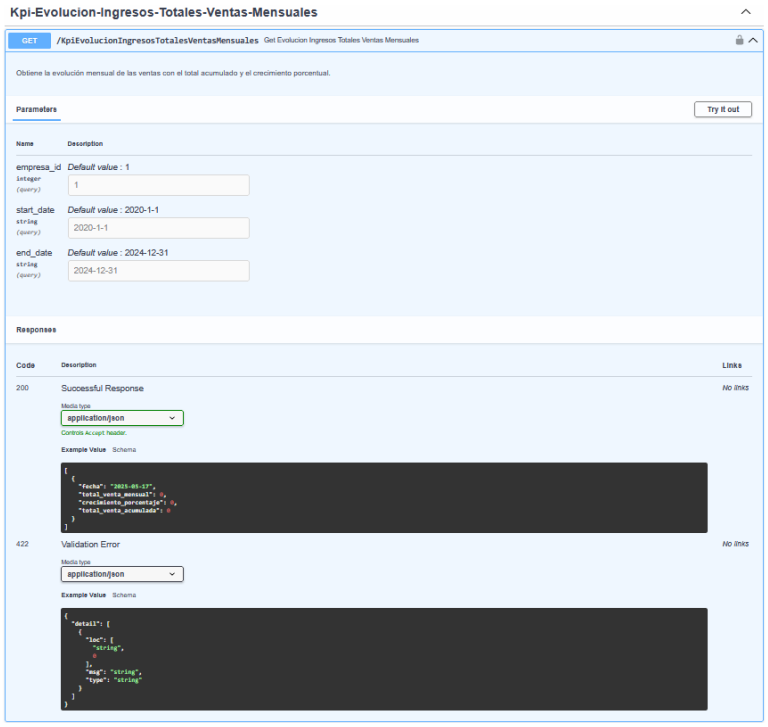


El indicador referenciado en la figura 84 permite analizar cómo varían los ingresos generados por ventas en cada mes a lo largo del tiempo. Su propósito es ofrecer una visión detallada y precisa del comportamiento financiero del negocio mes a mes, identificando tendencias, patrones estacionales y posibles fluctuaciones en el desempeño comercial. El seguimiento de este KPI facilita la evaluación del impacto de campañas, promociones, cambios de precios o condiciones del mercado en los resultados mensuales. Además, sirve como herramienta

clave para realizar ajustes oportunos en las estrategias de ventas y proyecciones financieras de corto y mediano plazo.

Figura 84

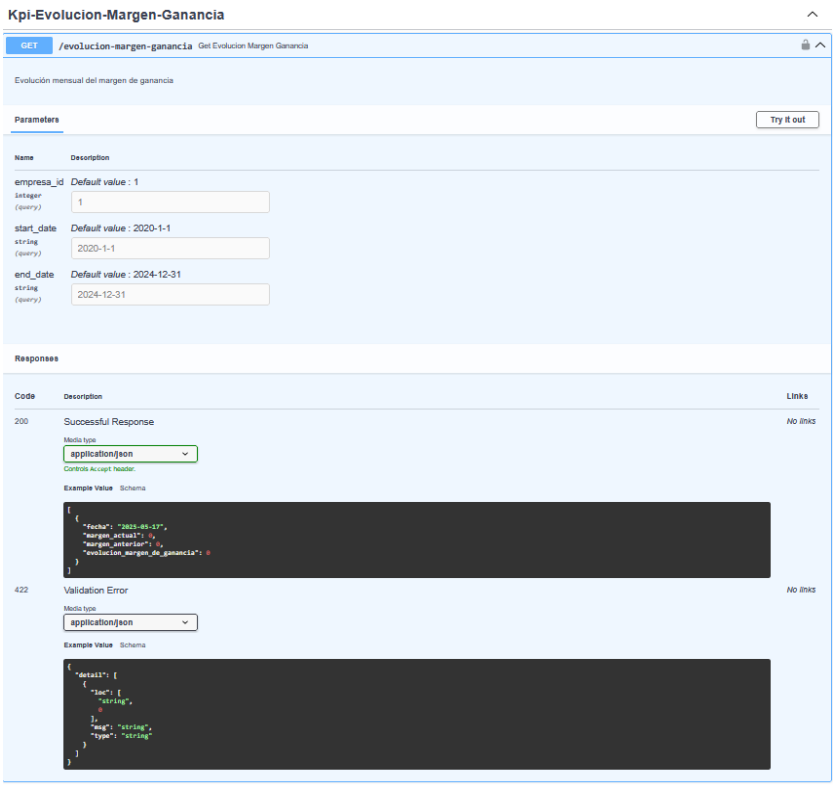
KPI de Evolución de Ingresos Totales por Ventas Mensuales



El indicador referenciado en la figura 85 analiza cómo ha variado el margen de ganancia de la empresa a lo largo del tiempo, permitiendo observar si la rentabilidad de las operaciones comerciales ha mejorado, se ha mantenido estable o ha disminuido. Se calcula mes a mes o trimestre a trimestre, comparando la utilidad obtenida con respecto a los ingresos totales por ventas. Su análisis es clave para entender el impacto de los costos, precios, estrategias de producto y eficiencia operativa sobre la rentabilidad. El seguimiento continuo de este KPI permite a la empresa tomar decisiones informadas sobre ajustes en precios, reducción de gastos o mejoras en procesos para maximizar el beneficio neto.

Figura 85

KPI de Evolución del Margen de Ganancia

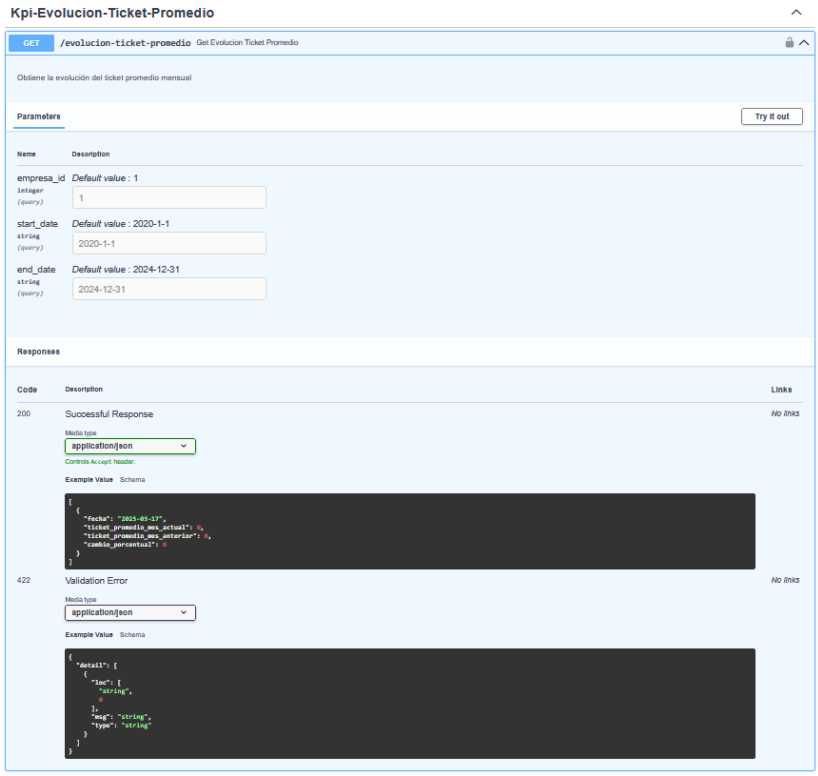


El indicador referenciado en la figura 86 muestra cómo varía el valor promedio de cada transacción en el tiempo, ya sea de forma mensual, trimestral o anual. Al analizar su evolución, es posible identificar tendencias en el comportamiento de compra de los clientes, evaluar el impacto de estrategias como ventas cruzadas, promociones o ajustes de precios, y detectar oportunidades para incrementar el valor de cada venta. El KPI de evolución del ticket promedio ayuda a

comprender si los clientes están gastando más o menos por compra con el paso del tiempo, lo cual es fundamental para mejorar la rentabilidad y orientar decisiones comerciales basadas en datos.

Figura 86

KPI de Evolución del Ticket Promedio

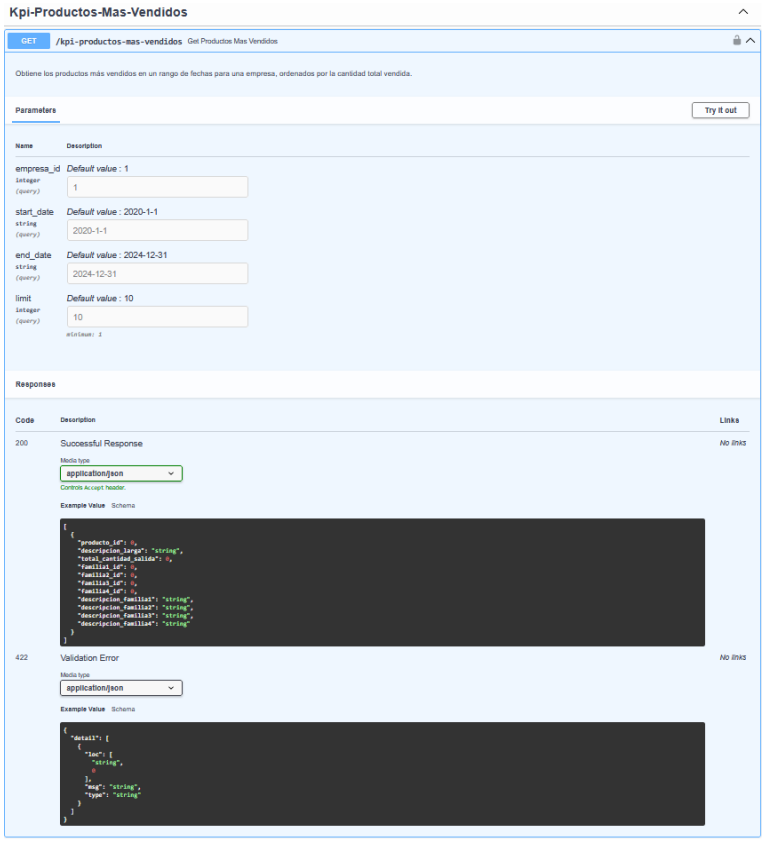


El indicador referenciado en la figura 87 identifica cuáles son los productos con mayor volumen de ventas en un periodo determinado, ya sea por cantidad de unidades vendidas o por ingresos generados. Su análisis permite conocer las preferencias de los clientes, optimizar el inventario, y enfocar estrategias de marketing y ventas en los artículos con mayor demanda. Además, este KPI es clave para la toma de decisiones sobre surtido de productos, promociones,

rotación de stock y desarrollo de nuevos productos. Evaluarlo periódicamente ayuda a maximizar la rentabilidad y a mantener alineada la oferta con el comportamiento del mercado.

Figura 87

KPI de Productos Más Vendidos



El indicador referenciado en la figura 88 permite identificar aquellos productos con menor rotación o demanda dentro de un periodo determinado, ya sea por volumen de unidades vendidas o por ingresos generados. Analizar los productos menos vendidos es fundamental para detectar posibles problemas en el portafolio, como artículos poco atractivos, precios inadecuados o falta de visibilidad. Este KPI también ayuda a tomar decisiones informadas sobre estrategias de liquidación, descontinuación de productos, ajustes de inventario y mejoras en la oferta comercial.

Su monitoreo regular contribuye a optimizar la eficiencia operativa y a mantener un catálogo alineado con las preferencias del mercado.

Figura 88

KPI de Productos Menos Vendidos

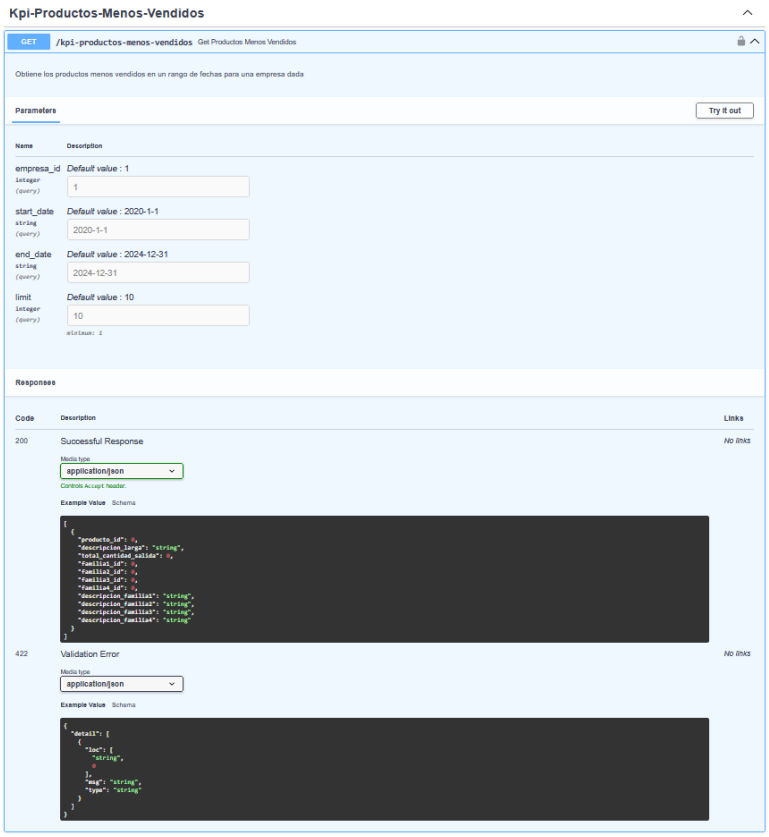


Tabla 29

Resumen de endpoints y características asociadas

Endpoint	KPI	Metodo HTTP	Story points	Dificultad
kpl-total-ventas	Ventas totales	GET	3	Media
precio-total-ventas	Precio promedio ventas	GET	2	Baja
utilidad-total-ventas	Utilidad neta	GET	4	Alta
kpl-margen-de-ganancia	Margen bruto	GET	3	Media

Evolucionventas-mensuales	Tendencia mensual	GET	5	Alta
ticket-promedio-mensual	Valor transacción promedio	GET	2	Baja
evolucion-ingresos-utilidad	Rentabilidad temporal	GET	4	Alta
kpl-Evolucion-Ingresos-TotaLevertaMensuales	Ingresos mensuales	GET	3	Media
Evolución-margen-ganancia	Evolución margen	GET	3	Media
Evolucion-ticket-promedio	Cambio valor ticket	GET	2	Baja
kpl-ventas-sucursal	Desempeño por ubicación	GET	5	Alta
kpl-productos-mas-vendidos	Productos Mas Vendidos	GET	1	Baja
kpl-productos-nenos-vendidos	Productos Menos Vendidos	GET	1	Baja
kpl-ventas-totales-anuales	Ventas Anuales	GET	2	Media
sucursales	Catálogo de tiendas	GET	1	Baja

5.6 Prototipos de Modelos Predictivos

En esta sección se describe el diseño e implementación de los modelos predictivos desarrollados para estimar las ventas futuras de productos a partir de datos históricos. El enfoque consistió en seleccionar modelos adecuados para series temporales, preparar los datos necesarios y evaluar el rendimiento de las predicciones obtenidas. Debido a que la plataforma está orientada a ofrecer predicciones por producto y por empresa, se optó por una solución escalable que permitiera entrenar múltiples modelos de forma eficiente.

A continuación, se expone el proceso completo desde la obtención, análisis y transformación de los datos, seguido por la explicación de los modelos seleccionados (ARIMA, SARIMA y Prophet).

5.6.1 Obtención, Análisis y Preparación de los Datos

Los datos utilizados para el diseño de los modelos se originaron a partir de una base de datos diseñada específicamente para el proyecto. Esta base fue alimentada mediante un proceso ETL que extrajo información desde la base de datos principal, transformando los datos en un formato adecuado para el análisis, como el total diario de productos vendidos.

Para facilitar el desarrollo experimental, se seleccionaron los tres productos con mayor volumen de ventas. Los registros asociados a estos productos fueron exportados a archivos en formato CSV, cada uno conteniendo información diaria con dos columnas principales:

1. FechaDeSalida: Fecha de la venta (YYYY-MM-DD).
2. TotalVendido: Número de unidades vendidas en el día correspondiente.

Los datos cubren un período continuo de 468 días (desde el 2 de enero del 2023 hasta el 13 de abril del 2024), lo que permite la construcción de series temporales con una cantidad suficiente de observaciones. Durante la revisión de calidad, se verificó que los archivos no contaran con valores nulos ni duplicados, que las fechas estuvieran ordenadas cronológicamente, y que algunas fechas no estuvieran presentes, como los festivos de navidad (25 de diciembre de 2023), año nuevo (1 de enero de 2024) y viernes santos (7 de abril de 2023 y 29 de marzo de 2024).

En cuanto a la periodicidad de los datos, se optó por trabajar con series diarias. Esta decisión se tomó debido a que, si se hubiera utilizado una frecuencia mensual, el total de observaciones disponibles habría sido considerablemente menor (aproximadamente 15 registros),

lo cual habría dificultado el entrenamiento de modelos estadísticos. La frecuencia diaria permitió disponer de una base más amplia para el análisis y modelado de series temporales.

A continuación, se presentan las gráficas con la distribución original de los datos por producto:

Figura 89

Ventas diarias del producto A

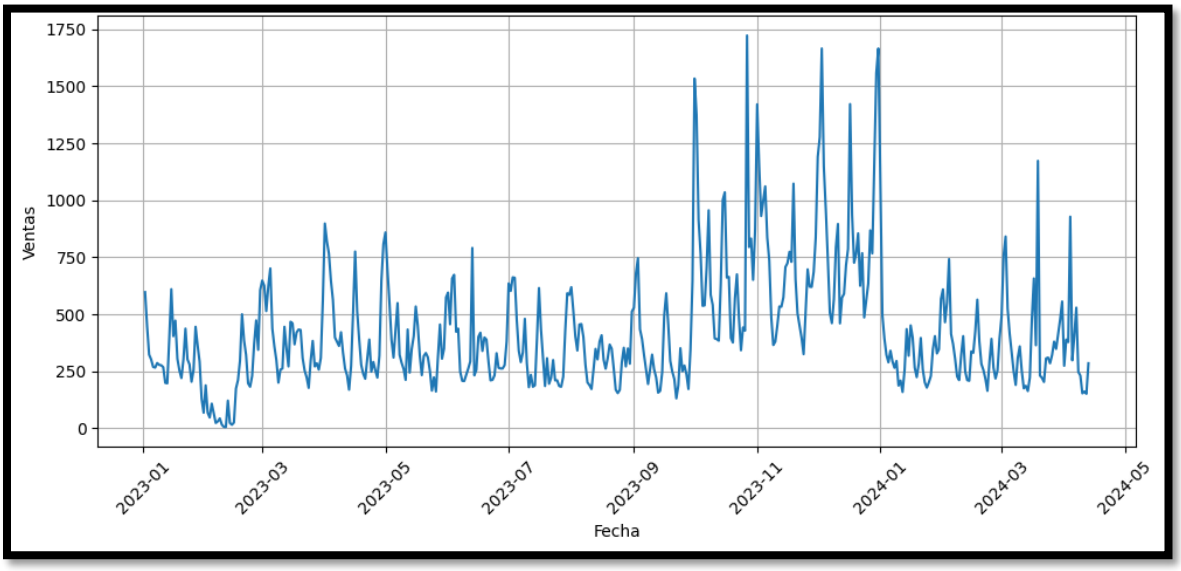


Figura 90

Ventas diarias del producto B

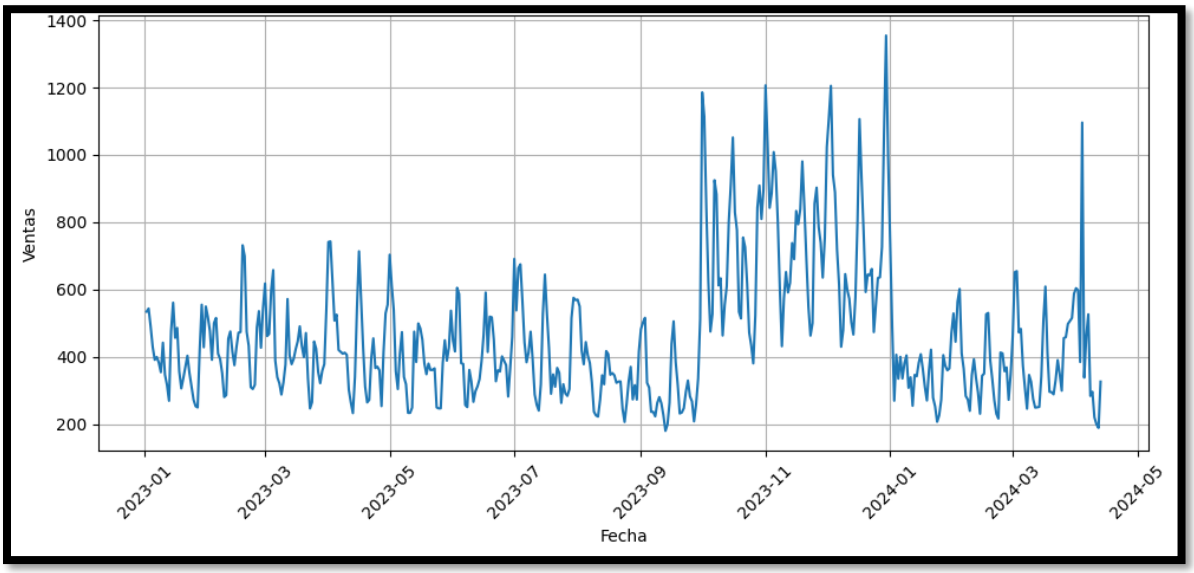
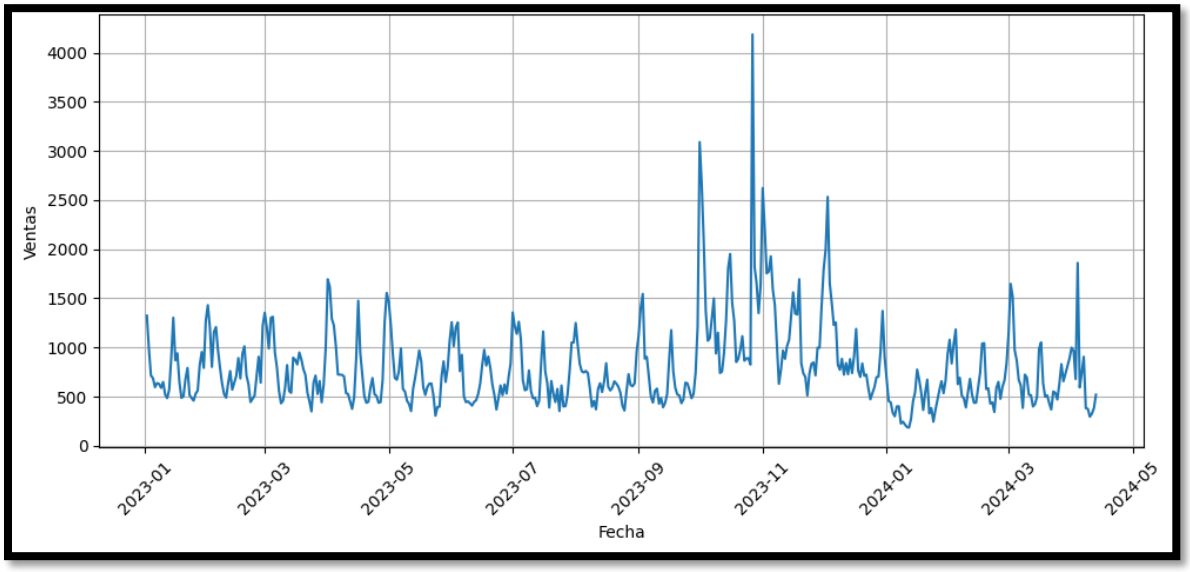


Figura 91

Ventas diarias del producto C



Antes del entrenamiento de los modelos, fue necesario preparar los datos para asegurar su consistencia y adecuación al análisis temporal. Las principales transformaciones realizadas fueron:

1. Conversión de tipo de datos: La columna de fechas se transformó a tipo datetime.

2. Normalización de nombres: Las columnas fueron renombradas como ds (fecha) y y (variable objetivo), según el formato requerido por algunos modelos.
3. Manejo de fechas faltantes: Se completaron manualmente las fechas ausentes, asignando un valor de 0, para mantener la continuidad temporal de la serie.

Estas transformaciones aseguran que las series sean compatibles con los modelos y preservan las características temporales relevantes.

A continuación, se muestran las gráficas con las series ya transformadas y listas para el entrenamiento:

Figura 92

Ventas diarias del producto A (Datos ordenados)

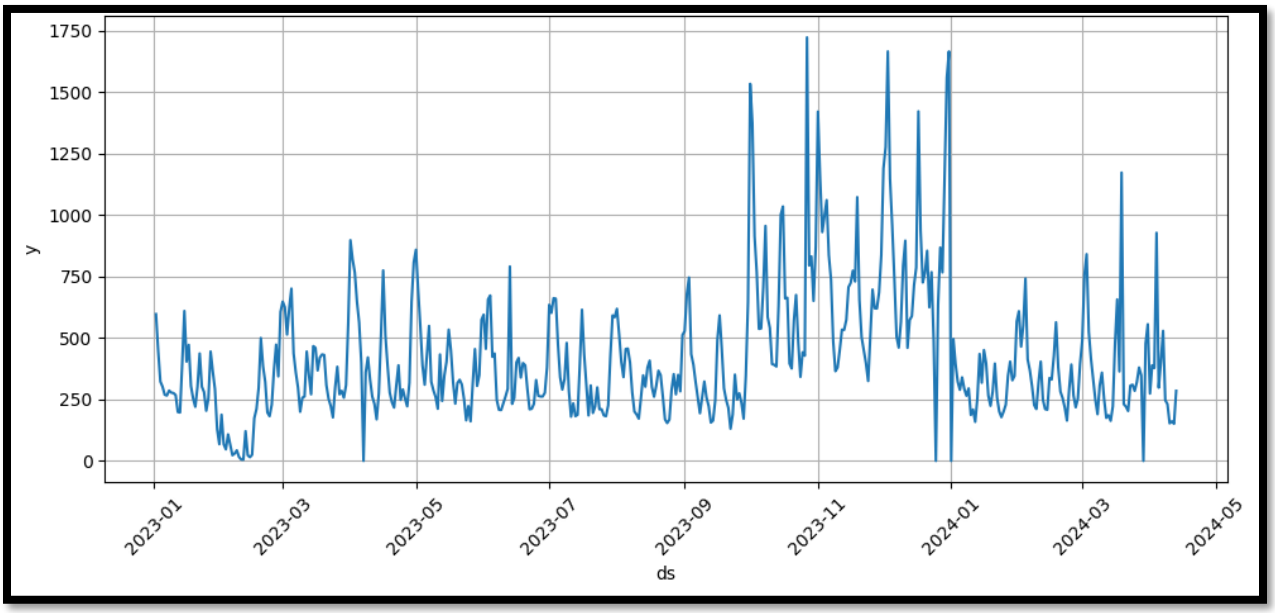


Figura 93

Ventas diarias del producto B (Datos ordenados)

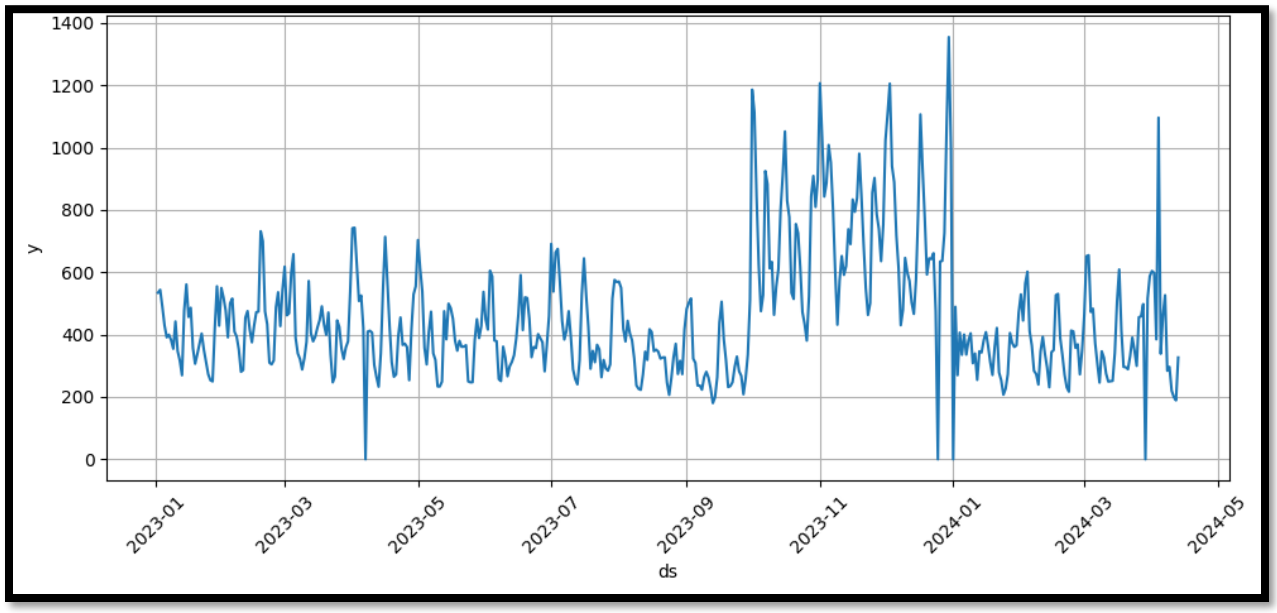
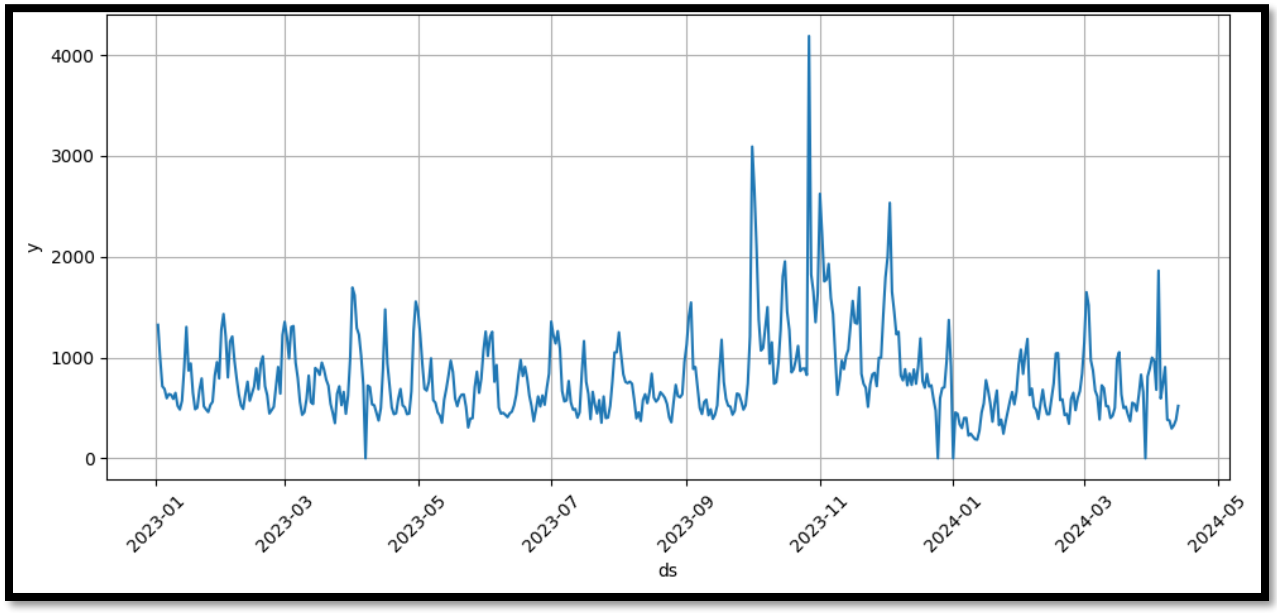


Figura 94

Ventas diarias del producto C (Datos ordenados)



5.6.2 Entrenamiento y Evaluación de los Modelos

Una vez preparados los datos y definidos los modelos a utilizar, se procedió con el entrenamiento y la evaluación de las predicciones para cada uno de los tres productos de forma individual. El objetivo principal de esta etapa fue evaluar la capacidad de los modelos para ajustarse a los patrones históricos de ventas y predecir el comportamiento futuro.

Para cada producto, los datos se dividieron en un conjunto de entrenamiento y otro de prueba. El conjunto de entrenamiento, que representó el 80 % de los datos (equivalente a aproximadamente 12 meses de ventas), se utilizó para ajustar el modelo. El conjunto de prueba, correspondiente al 20 % restante (alrededor de 3 meses), se empleó para evaluar su desempeño sobre datos no vistos previamente.

5.6.2.1 Criterios metodológicos para la selección del modelo

En la fase de modelado se evaluaron distintas alternativas con el propósito de identificar el enfoque más adecuado para cada producto. Inicialmente se exploraron modelos ARIMA tradicionales; sin embargo, sus resultados fueron poco satisfactorios debido a que no lograron capturar adecuadamente los patrones de estacionalidad presentes en las series. Por esta razón, se optó por trabajar con SARIMA, una extensión que incorpora componentes estacionales y permite una mejor representación de la dinámica de los datos. Paralelamente, se evaluó el desempeño del modelo Prophet, cuya facilidad de implementación y capacidad para manejar tendencias y estacionalidades complejas resultó de interés en este estudio.

El proceso para seleccionar el mejor modelo SARIMA siguió la metodología Box-Jenkins, apoyada en técnicas de automatización:

1. En primer lugar, se evaluó la estacionariedad de la serie mediante la prueba de Dickey-Fuller Aumentada (ADF). Si la hipótesis nula no era rechazada, se aplicaron diferenciaciones hasta lograr estacionariedad, lo que permitió determinar el valor de d.
2. Dado que las series mostraban patrones repetitivos en intervalos específicos, se asumió un componente estacional con D = 1.
3. Los parámetros (p, q, P, Q) se estimaron automáticamente con la función auto_arima de pmdarima, la cual itera y prueba diversas combinaciones hasta encontrar la especificación más adecuada para un valor dado de m (periodicidad estacional). En este trabajo se probaron distintas configuraciones con m = 7, 14, 15, 16, 28, 30 y 31.

Como criterio inicial de comparación entre modelos, se utilizó el Akaike Information Criterion (AIC), que busca un equilibrio entre precisión y complejidad: valores más bajos de AIC indican un mejor ajuste con menor sobreajuste. No obstante, dado que el AIC no refleja directamente la calidad predictiva, la selección final se basó en métricas de error en el conjunto de prueba: MAE (Mean Absolute Error), RMSE (Root Mean Squared Error) y MAPE (Mean Absolute Percentage Error). Estas métricas permiten evaluar la magnitud y el impacto relativo de los errores de predicción, ofreciendo una medida más robusta del rendimiento.

En algunos casos, los modelos con menor AIC no fueron necesariamente los que presentaron mejores valores de error. En tales situaciones se priorizó la precisión predictiva sobre el criterio de información, seleccionando aquellos modelos que ofrecieron menores errores en MAE, RMSE y MAPE.

Para Prophet, el proceso fue más directo: se ajustó el modelo a la serie con los parámetros yearly_seasonality = True y country_name = 'Colombia', lo que permitió incorporar tanto la estacionalidad anual como los días festivos nacionales de manera automática.

Finalmente, la decisión entre el mejor modelo SARIMA y Prophet para cada producto se realizó mediante la comparación directa de sus métricas de error en los datos de prueba. El modelo con mejor desempeño predictivo en cada caso fue el que se seleccionó como modelo final.

5.6.2.2 Modelo seleccionado por producto

Con base en los criterios metodológicos definidos, se compararon los resultados del mejor modelo SARIMA frente a Prophet para cada producto. La Tabla 24 presenta un resumen de los indicadores de desempeño en el conjunto de prueba:

Tabla 30

Comparación de resultados SARIMA vs Prophet por producto

Producto	Modelo	MAE	RMSE	MAPE	% dentro del intervalo
Producto A	SARIMA (m=31)	139.01	179.03	43.22%	100%
	Prophet	166.5	206.87	56.06%	80.85%
Producto B	SARIMA (m=31)	119.89	156.32	33.29%	100%
	Prophet	90.46	126.26	26.01%	91.45%
Producto C	SARIMA (m=31)	283.16	365.02	43.71%	98.94%
	Prophet	267.03	372.33	35.44%	80.85%

A continuación, se presentarán los análisis de los resultados obtenidos:

1. Producto A: El modelo SARIMA (m=31) presentó consistentemente mejores métricas que Prophet, con menores valores de MAE, RMSE y MAPE, además de un 100% de observaciones dentro de los intervalos de predicción. Esto evidencia una mayor estabilidad y confiabilidad del modelo, por lo cual se selecciona como el modelo final para este producto.

2. Producto B: En este caso, el modelo Prophet superó a SARIMA en todas las métricas de error, con reducciones importantes en MAE (−29.43) y MAPE (−7.28%). Aunque SARIMA alcanzó un cubrimiento total en los intervalos, la precisión global de Prophet fue superior, por lo cual se selecciona este modelo como el más adecuado.
3. Producto C: Los resultados muestran un escenario más equilibrado. SARIMA obtuvo un mejor RMSE y un porcentaje de cubrimiento casi total (98.94%), mientras que Prophet alcanzó menores valores de MAE y MAPE. Dado que el objetivo principal era minimizar el error medio y relativo, se priorizó el desempeño de Prophet, a pesar de su menor cobertura de intervalos.

Finalmente, en las Figuras 95, 96 y 97 se presentan los gráficos de predicción correspondientes a los modelos seleccionados para cada producto. Estos permiten visualizar el ajuste logrado frente a los valores reales y la capacidad del modelo de capturar la tendencia y estacionalidad de la serie.

Figura 95

Predicción Sarima Producto A

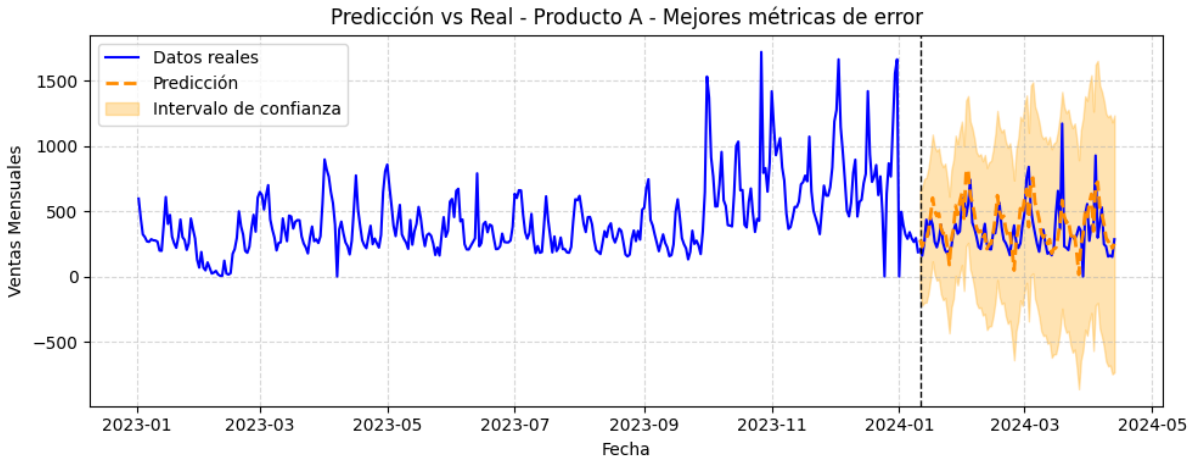


Figura 96

Predicción Prophet Producto B

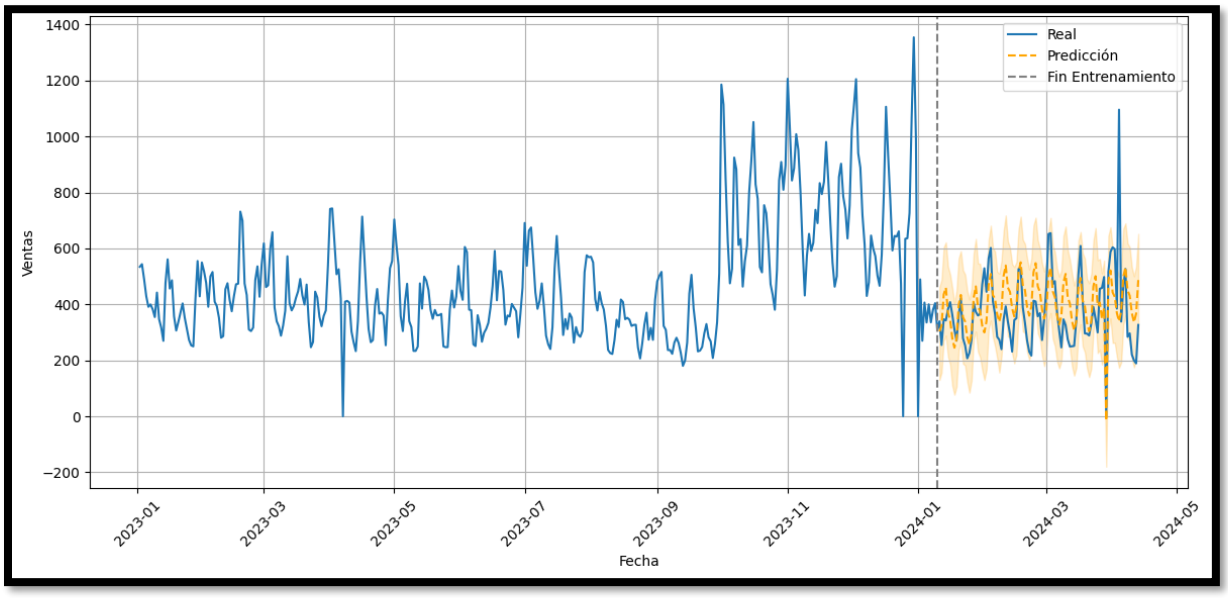
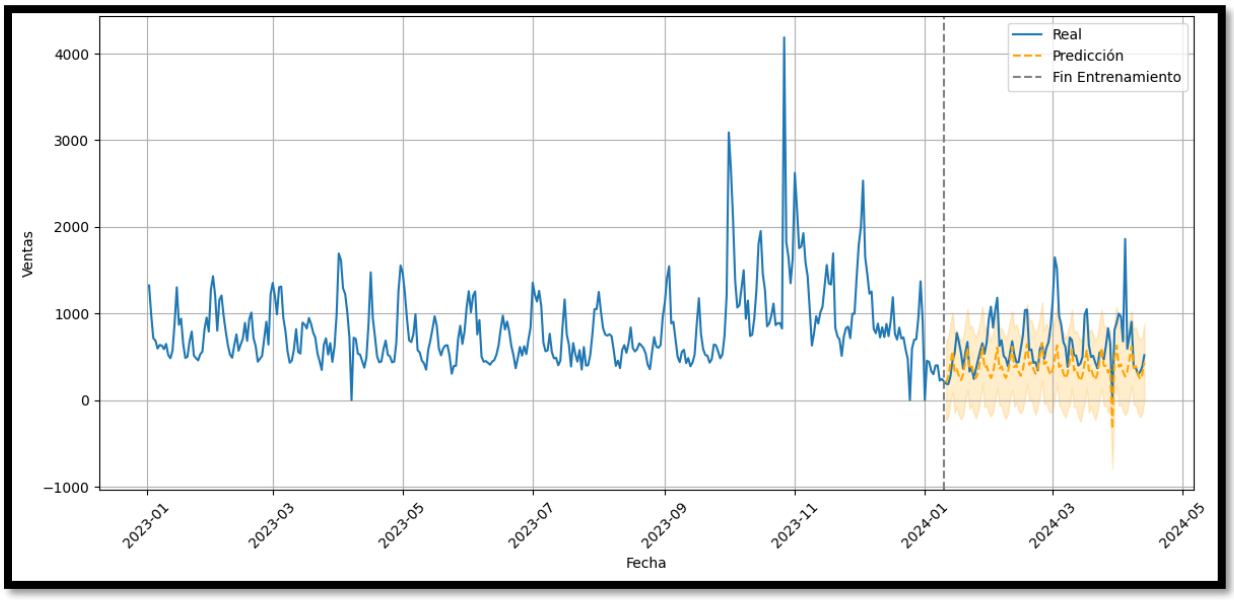


Figura 97

Predicción Prophet Producto C



Cabe destacar que, aunque Prophet presentó mejores métricas de error en dos de los tres productos, los resultados gráficos evidencian una diferencia importante en la forma en que cada

modelo representa la serie. Las predicciones de Prophet tienden a ubicarse en valores centrales, lo que genera errores absolutos relativamente bajos, pero con una pérdida de capacidad para capturar los picos y variaciones más pronunciadas de la serie. En contraste, SARIMA logra representar de manera más fiel la tendencia y los patrones estacionales de los datos, aunque sus predicciones en ciertos puntos extremos puedan distanciarse más de los valores reales. Este aspecto debe considerarse en la interpretación de los resultados y en la elección del modelo en función del objetivo práctico del análisis: minimizar errores medios o reflejar de forma más precisa la dinámica de la serie.

5.6.2.3 Lineamientos para la elección del modelo

El análisis comparativo realizado muestra que tanto SARIMA como Prophet tienen ventajas específicas que hacen más adecuada su aplicación en diferentes escenarios.

En el contexto del sector retail colombiano, donde cada empresa maneja un gran número de productos, la implementación de Prophet puede resultar más eficiente. Su facilidad de configuración y ajuste automático de parámetros reduce el tiempo necesario para entrenar un modelo por producto, lo que lo convierte en una opción práctica cuando se trabaja con múltiples series de demanda.

Por otro lado, SARIMA resulta especialmente útil cuando se busca un modelo más interpretativo y ajustado a la dinámica de la serie. En los tres productos analizados, el modelo con periodicidad mensual (m = 31) mostró un buen desempeño, lo que sugiere que series con características similares podrían modelarse eficazmente bajo esta configuración. Sin embargo, en productos con menor rotación o con alta proporción de valores en cero, este enfoque puede no ser adecuado, requiriendo ajustes adicionales en los parámetros o incluso la exploración de modelos alternativos.

Asimismo, para mejorar la suavización de las series y reducir el ruido presente en datos diarios, resulta recomendable trabajar con escalas temporales más amplias, como series mensuales. Esta agregación no solo estabiliza los patrones, sino que también facilita la identificación de tendencias y ciclos de manera más clara.

En síntesis, Prophet es preferible en escenarios donde prima la escalabilidad y rapidez de implementación, mientras que SARIMA ofrece un mayor nivel de detalle y capacidad de representar la estacionalidad cuando se dispone de series con comportamientos más regulares. La elección entre ambos dependerá, por tanto, del equilibrio entre la necesidad de eficiencia operativa y la profundidad analítica requerida.

5.7 Pruebas

Durante el proceso de desarrollo se desplegó un servidor de pruebas para cada entorno (frontend y backend), con el objetivo de implementar distintos tipos de pruebas y verificar el correcto funcionamiento de las funcionalidades a medida que eran desarrolladas, asegurando el cumplimiento de los requerimientos establecidos y una experiencia de usuario estable.

5.7.1 Pruebas en el frontend

Para la fase de pruebas relacionadas con las vistas e interfaces de usuario (frontend) se implementaron pruebas unitarias y de componentes, utilizando Vitest como entorno principal de pruebas, con el objetivo de verificar el correcto funcionamiento de la interfaz de usuario y favorecer una experiencia estable para el usuario final.

5.7.1.1 Pruebas unitarias

Se utilizó la librería Jest para facilitar la creación de pruebas más específicas, centradas en funciones individuales generales del proyecto. Estas pruebas permitieron validar pequeñas unidades de código de forma aislada, asegurando su correcto comportamiento ante distintos escenarios.

5.7.1.2 Pruebas de componentes

Se utilizó Testing Library para realizar pruebas centradas en el comportamiento de los componentes. A través de esta herramienta, se simularon interacciones reales, como clics, ingreso de datos y navegación; para validar que los componentes respondieran adecuadamente. También se evaluaron distintos estados de la interfaz, tales como cargas, errores y respuestas exitosas, lo que permitió ofrecer una experiencia de usuario robusta.

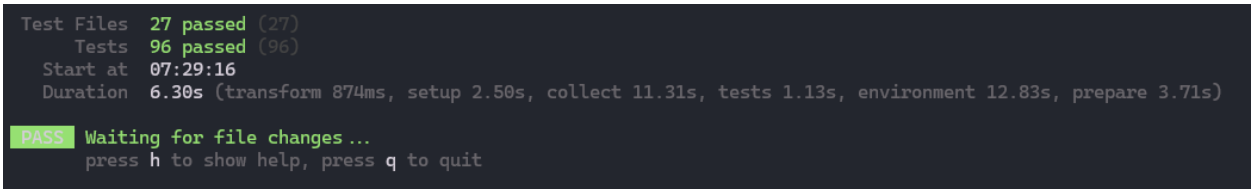
5.7.1.3 Resultados de las pruebas

El desarrollo de las pruebas realizadas junto a las herramientas mencionadas permitió detectar y corregir inconsistencias en etapas tempranas del desarrollo frontend. Resultando en el correcto comportamiento de las múltiples funcionalidades que pasaron por este proceso, manteniendo la estabilidad y fiabilidad de la interfaz de usuario.

En la figura 117 se muestra el resumen de las 96 pruebas automatizadas ejecutadas, todas aprobadas sin errores, distribuidas en 27 archivos. El proceso tomó 6.30 segundos y el entorno quedó en espera activa para detectar cambios y ejecutar pruebas automáticamente.

Figura 98

Resultados de pruebas de frontend



5.7.2 Pruebas en el Backend

Con el objetivo de verificar el correcto funcionamiento del software desarrollado, se realizaron pruebas tanto unitarias como de integración, cubriendo las principales capas de la arquitectura del backend.

5.7.2.1 Pruebas de integración: Repositorio - Base de datos

Las pruebas de integración se centraron en la capa de infraestructura, específicamente en los repositorios, los cuales interactúan directamente con la base de datos mediante consultas SQL.

Para realizar estas pruebas, se creó una base de datos de prueba aislada del entorno de producción. Cada prueba se estructuró de la siguiente forma:

1. Preparación: antes de ejecutar el código a probar, se insertaban en la base de datos los datos necesarios para la prueba en cuestión (por ejemplo, empresas, productos o usuarios).
2. Ejecución: se ejecutaba el método del repositorio y se verificaba su comportamiento con distintos tipos de entradas.
3. Verificación: se aplicaban instrucciones assert para confirmar que los resultados fueran correctos y esperados.

4. Limpieza: al final de cada prueba se eliminaban los datos insertados previamente, mediante sentencias DELETE, para evitar efectos secundarios entre pruebas y mantener la base de datos limpia.

Este enfoque permitió probar la funcionalidad real de los repositorios y asegurar que la lógica de acceso a datos estuviera correctamente implementada y sincronizada con el esquema de la base de datos.

5.7.2.2 Pruebas unitarias: Servicios de la capa de aplicación

Las pruebas unitarias se enfocaron en la capa de aplicación, específicamente sobre los servicios que contienen la lógica de negocio del software. Para cada servicio se diseñaron múltiples casos de prueba con diferentes entradas, de forma que se evaluaran tanto escenarios típicos como bordes y posibles errores.

Dado que estos servicios dependen de repositorios y, en algunos casos, de otros servicios, se utilizó la librería Moq para mockear las dependencias. Esto permitió aislar la lógica de negocio y verificar que los servicios reaccionaran correctamente ante diversos escenarios simulados.

Este enfoque aseguró que los servicios se comportaran correctamente sin necesidad de conectarse a una base de datos real, haciendo las pruebas más rápidas, repetibles y centradas exclusivamente en la lógica propia del servicio.

5.7.2.3 Pruebas manuales de los endpoints

Además de las pruebas automatizadas, se realizaron pruebas manuales a los distintos endpoints del backend, con el fin de verificar su correcto funcionamiento desde la perspectiva de un cliente que consume la API. Estas pruebas permitieron evaluar el software de forma más

integral, incluyendo aspectos como el formato de las respuestas, el manejo de errores y la lógica general de interacción.

Para llevar a cabo estas pruebas se utilizaron herramientas como Postman y Swagger UI, que facilitaron la ejecución de peticiones HTTP con distintos métodos (GET, POST, PUT, DELETE) y parámetros. Se probaron múltiples escenarios, entre ellos:

1. Solicitudes válidas con parámetros correctos, verificando que los datos retornados coincidieran con lo esperado.
2. Solicitudes inválidas con parámetros erróneos o faltantes, confirmando que el software respondiera con los códigos de estado HTTP adecuados (por ejemplo, 400 Bad Request, 404 Not Found o 500 Internal Server Error).
3. Validación de respuestas vacías, formatos de fechas, listas y objetos anidados.
4. Casos límite como fechas fuera de rango o combinaciones de empresa y producto inexistentes.

Estas pruebas fueron fundamentales para asegurar que el software no solo funciona correctamente desde el punto de vista interno, sino también desde su interfaz pública, brindando una experiencia consistente, robusta y segura a los consumidores de la API.

5.7.2.4 Resultados de las pruebas

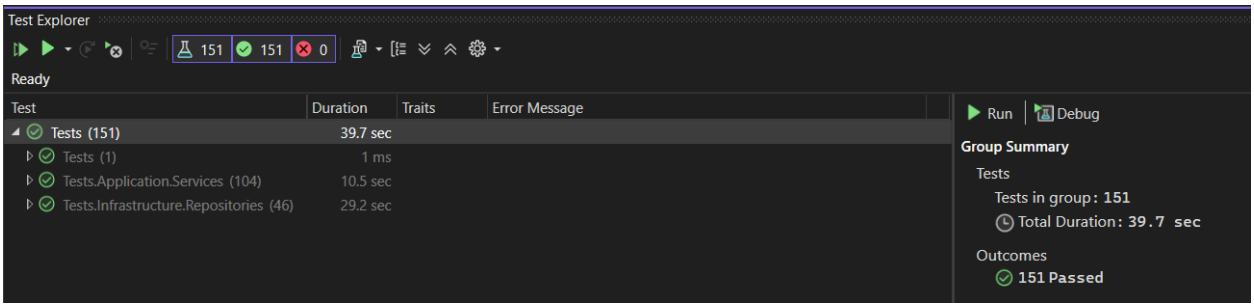
Todas las pruebas realizadas, tanto unitarias, de integración, como manuales, permitieron validar el correcto funcionamiento del software desde distintas capas. Las pruebas unitarias aseguraron que la lógica de negocio se comportara adecuadamente ante distintos escenarios, mientras que las pruebas de integración confirmaron que los repositorios interactúan correctamente con la base de datos.

Por otro lado, las pruebas manuales aplicadas a los endpoints de la API permitieron verificar de forma directa las respuestas del software ante diferentes entradas. Gracias a estas pruebas se pudo confirmar que los servicios expuestos funcionan de manera estable y coherente en un entorno real.

Todas las pruebas unitarias y de integración fueron ejecutadas utilizando Visual Studio. La Figura 118 muestra un resumen detallado de las 151 pruebas automatizadas ejecutadas, todas con resultado exitoso y sin errores. Estas pruebas, distribuidas entre distintos módulos del sistema, se completaron en un tiempo total de 39.7 segundos, lo que evidencia la estabilidad y correcto funcionamiento del software evaluado.

Figura 99

Resultados de las pruebas del backend



6 Conclusiones

El desarrollo de una plataforma web para la toma de decisiones en el sector retail colombiano constituyó un ejercicio integral de ingeniería de software, donde convergieron diseño arquitectónico, análisis de datos, desarrollo de interfaces, y métodos ágiles de gestión de proyectos, y por tanto se expresan las siguientes conclusiones:

1. En el aspecto metodológico, se adoptó un enfoque híbrido basado en Scrumban y prácticas de Extreme Programming (XP). Esto permitió combinar la planificación estructurada de Scrum con la flexibilidad visual de Kanban, facilitando la priorización de tareas, la entrega continua y la adaptación a cambios. Además, las prácticas de XP como la refactorización constante, el diseño simple y los estándares de codificación contribuyeron significativamente a la calidad del producto final.
2. Desde el punto de vista técnico, el backend desarrollado con .NET y C# para la gestión empresarial, junto con un backend en Python y FastAPI para análisis y predicción, permitió la separación de responsabilidades; Este diseño modular favorecerá la escalabilidad y mantenibilidad del software.
3. En lo referente a la experiencia de usuario, se diseñaron e implementaron interfaces modernas utilizando tecnologías como React, Tailwind CSS y Shadcn/ui, las cuales ofrecieron una experiencia fluida, responsiva y accesible. Las vistas desarrolladas contemplaron desde formularios de autenticación hasta dashboards interactivos de KPI, incluyendo soporte para modo claro/oscuro, adaptabilidad móvil, y una navegación intuitiva, enfocándose en la usabilidad y la eficiencia visual.

4. La arquitectura de datos implementada, sustentada en un modelo relacional con soporte multicliente, fue complementada con un proceso ETL robusto que aseguró la integridad, transformación y disponibilidad de los datos operacionales y analíticos.
5. En el ámbito predictivo, se aplicaron y evaluaron modelos de series temporales.
    a. Modelo Arima

        El modelo ARIMA fue utilizado como punto de partida para las pruebas de predicción. Este modelo no incluye componentes estacionales, por lo que su aplicación en un contexto de ventas diarias, donde pueden existir variaciones cíclicas por día de la semana o del mes, no permitió capturar ciertos patrones presentes en las series. Esto limitó la capacidad del modelo para ajustarse a los datos observados.

    b. Modelo Sarima

        El modelo SARIMA fue aplicado para incorporar la estacionalidad presente en las series de ventas diarias. Este enfoque permitió capturar mejor los patrones que se repiten a lo largo del tiempo, especialmente en productos con frecuencia alta de venta. Su aplicación se centró en ajustar los modelos a las características de cada producto, considerando las variaciones cíclicas identificadas durante el análisis exploratorio. Dado que las series trabajadas correspondían a productos con ventas constantes, este tipo de modelo puede ser considerado en contextos similares, mientras que para productos con menor frecuencia de venta su desempeño deberá evaluarse según el comportamiento de cada caso.

    c. Modelo Prophet

        El modelo Prophet fue utilizado con el objetivo de capturar tanto la tendencia general como los efectos estacionales anuales y de días festivos en las series de ventas. Su configuración incluyó la incorporación del calendario oficial de festivos en Colombia, lo cual permitió

considerar posibles impactos de estas fechas en la demanda. El modelo fue entrenado individualmente para cada producto, logrando resultados adecuados en productos con comportamiento regular y patrones definidos. Si bien logró representar de forma clara la tendencia y ciertos ciclos estacionales, su precisión disminuyó en periodos de alta variabilidad, posiblemente influenciados por factores no modelados como promociones o ajustes de precio. Este tipo de modelo resulta apropiado para contextos donde se desea interpretar componentes temporales de manera comprensible, especialmente en productos con estacionalidad moderada y suficiente historial de datos. Sin embargo, su desempeño puede verse limitado en productos con patrones altamente irregulares o con eventos comerciales inesperados.

d. Comparación entre modelos Sarima y Prophet.

   La comparación entre ambos modelos mostró que Prophet obtuvo mejores métricas de error en dos de los tres productos analizados (B y C), mientras que SARIMA fue superior en el producto A, además de ofrecer una mejor representación de la tendencia y los picos de la serie en general. Esto sugiere que Prophet tiende a producir predicciones más suavizadas y cercanas al promedio de los datos, lo que reduce los errores absolutos, pero a costa de no reflejar con la misma precisión los valores extremos. SARIMA, en contraste, aunque puede requerir mayor trabajo en la identificación de parámetros y ajuste de la estacionalidad, permite capturar de manera más fiel la dinámica interna de la serie.

   En consecuencia, la elección del modelo depende tanto del objetivo del análisis como del contexto. Si la prioridad es la escalabilidad y facilidad de implementación —como en escenarios del sector retail con un gran número de productos— Prophet resulta más

eficiente. Por otro lado, si el interés radica en comprender y representar con detalle las variaciones estacionales y la tendencia real de la serie, SARIMA se convierte en una alternativa más robusta, siempre que se cuente con los recursos necesarios para su ajuste y mantenimiento.

6. Las pruebas funcionales, de integración y unitarias confirmaron el cumplimiento de los requerimientos establecidos, asegurando la calidad, estabilidad y rendimiento de la plataforma.

En suma, este proyecto demostró que es posible integrar de manera efectiva tecnologías actuales de desarrollo, metodologías ágiles y técnicas de análisis predictivo para construir una solución innovadora y práctica que contribuya formalmente a la transformación digital del sector retail colombiano.

7 Recomendaciones Futuras

A partir del desarrollo realizado y los resultados obtenidos, se identificaron algunos aspectos que podrían ser considerados en trabajos posteriores para ampliar las capacidades del software, optimizar su rendimiento o incorporar nuevas funcionalidades. A continuación, se presentan una serie de recomendaciones que surgen como posibles líneas de mejora o extensión del proyecto en futuras fases de desarrollo.

1. Aumentar la cantidad y calidad de los datos históricos: Uno de los principales factores que limitó la efectividad de ambos modelos fue la escasez o baja granularidad de datos históricos. Se recomienda integrar fuentes de datos adicionales o extender el periodo de recolección para obtener series más completas, lo que podría mejorar significativamente la capacidad predictiva de los modelos implementados.
2. Optimizar el consumo de modelos desde el backend principal: Actualmente el backend de gestión se comunica con el servicio de predicción para acceder a los modelos entrenados. Una futura mejora podría incluir la implementación de colas de tareas (como Celery o RabbitMQ) para manejar peticiones de predicción en segundo plano, especialmente si el número de usuarios o peticiones crece.
3. Monitoreo y reentrenamiento periódico de modelos: Se recomienda establecer un software que permita de manera automática detectar cuándo un modelo entrenado comienza a perder precisión (por ejemplo, mediante un umbral de error) y desencadene un reentrenamiento con datos actualizados, asegurando así una predicción continua y confiable.
4. Optimización del proceso ETL y consultas SQL: Actualmente, la carga de datos hacia el software de predicción puede resultar lenta debido a la complejidad de las consultas,

las cuales incluyen múltiples joins entre tablas. Sería recomendable estudiar técnicas especializadas de optimización, como la creación de vistas materializadas, el uso de índices adecuados, particionado de tablas, o incluso herramientas externas de orquestación ETL que permitan paralelizar procesos o realizar transformaciones previas fuera del motor de base de datos principal.

5. Documentación y escalabilidad del prototipo: Finalmente, para facilitar la evolución del software, es recomendable mantener una documentación técnica actualizada y modularizar aún más los componentes, de modo que cada servicio pueda evolucionar de forma independiente y escalar sin fricciones.

Referencias Bibliográficas

1Kosmos. (18 de Abril de 2023). *¿Qué es la Autenticación JWT? ¿Cómo Funciona?* https://www.1kosmos.com/authentication/jwt-authentication/

Actian. (11 de Noviembre de 2022). *¿Qué es SQL Server?* https://www.actian.com/what-is-sql-server/

Allotey, C. (13 de Enero de 2025). *Getting Started with Vitest for Vue.js and Vite Testing*. Vue School Articles. https://vueschool.io/articles/vuejs-tutorials/start-testing-with-vitest-beginners-guide/

Alonso, A. (23 de Diciembre de 2018). *Enfoques de Arquitectura Multitenant para Aplicaciones SaaS*. Medium. https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

AltexSoft Editorial Team. (18 de Enero de 2021). *Extreme Programming: Values, Principles, and Practices*. https://www.altexsoft.com/blog/extreme-programming-values-principles-and-practices/

Arias, J. F., Reyna, B. D., & Mamani, G. (2021). Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android. *TecnoLógicas, 24*(52), 226-241. https://doi.org/10.22430/22565337.2104

AWS. (8 de Julio de 2022a). *¿Qué es el análisis de datos? - Explicación del análisis de datos - AWS*. Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/data-analytics/

AWS. (18 de Octubre de 2022b). *¿Qué es extracción, transformación y carga (ETL)?* https://bit.ly/aws-que-es-etl.

AWS. (13 de Septiembre de 2022c). *¿Qué es NET?* https://aws.amazon.com/es/what-is/net/

Axios. (s.f.). *Getting started | Axios Docs*. https://axios-http.com/docs/intro

Bautista, E. A., Valles, J. J., Collantes, H., Casildo, N. E., & Sánchez, J. E. (2024). Arquitectura multi-tenant para la implementación de un software como servicio y su influencia en la usabilidad de las MYPES del sector comercio del nororiente Peruano. *Ingeniare. Revista Chilena de Ingeniería, 32*. https://dx.doi.org/10.4067/s0718-33052024000100205

Besoftware. (5 de Mayo de2020). *¿Qué es C# y para qué sirve?* https://bsw.es/que-es-c/

Calderón, N. (7 de Marzo de 2021). *Entendiendo a la arquitectura limpia*. Medium. https://nescalro.medium.com/entendiendo-a-la-arquitectura-limpia-7877ad3a0a47

Cambarieri, M.G., Difabio, F., & García, N. (2020). Implementación de una arquitectura de software guiada por el dominio. *XXI Simposio Argentino de Ingeniería de Software (ASSE 2020) - JAIIO 49*, (pp. 192-209). Modalidad virtual. http://sedici.unlp.edu.ar/handle/10915/115198

Code Maze. (19 de Septiembre de 2023). *How to Use Moq to Return a Value That Was Passed Into a Method?* https://code-maze.com/dotnet-use-moq-return-value-that-was-passed-into-a-method/

Code Maze. (31 de Enero de 2024). *How to Secure Passwords with BCrypt.NET*. https://code-maze.com/dotnet-secure-passwords-bcrypt/

Cursor. (s.f.). *Features | Cursor - the AI Code Editor*. https://www.cursor.com/features

Datacamp. (2 de Septiembre de 2024). *Cursor AI: A Guide With 10 Practical Examples*. Datacamp. https://www.datacamp.com/tutorial/cursor-ai-code-editor

Datamind. (13 de Junio de 2023). *Las principales desventajas de tener mucha información, pero sin ningún tipo de análisis de datos*. https://godatamind.com/blog/las-principales-desventajas-de-tener-mucha-informacion-pero-sin-ningun-tipo-de-analisis-de-datos/

Datascientest. (19 de Diciembre de 2024). *Pandas : La biblioteca de Python dedicada a la Data Science*. https://datascientest.com/es/pandas-python

ESLint. (s.f.). *Core Concepts - ESLINT - Pluggable JavaScript Linter*. https://eslint.org/docs/latest/use/core-concepts/

FastApi. (s.f.). FastAPI. https://fastapi.tiangolo.com/

GeeksforGeeks. (7 de octubre de 2024). *Introduction to Tailwind CSS*. https://www.geeksforgeeks.org/introduction-to-tailwind-css/

Goncalves, L. (26 de Julio de 2024). *95% de las empresas en Latinoamérica lucha con la gestión de datos, según estudio de Dell Technologies*. Forbes Centroamérica. https://forbescentroamerica.com/2024/07/26/95-de-las-empresas-en-latinoamerica-lucha-con-la-gestion-de-datos-segun-estudio-de-dell-technologies

Google Fonts. (2025). *Geist*. https://fonts.google.com/specimen/Geist/about

Google. (s.f.). *Analytics*. Google Marketing Platform. https://marketingplatform.google.com/intl/es/about/analytics/

Jenkins. (2013). *Modelación ARIMA.* http://www.ptolomeo.unam.mx:8080/jspui/bitstream/132.248.52.100/363/7/A7.pdf

JWT. (5 de Octubre de 2015). *Introducción a los Tokens Web JSON*. https://jwt.io/introduction

Kalinda, C. (29 de Octubre de 2023). *Moq C# (Cómo funciona para desarrolladores)*. Iron Pdf. https://bit.ly/ironpdf-que-es-moq.

Kanban Tool. (6 de Agosto de 2024). *¿Qué es Scrumban?* https://kanbantool.com/es/guia-kanban/que-es-scrumban

Kosaka, M. (5 de Enero de 2021). *Efficient Time-Series Using Python's Pmdarima Library*. Towards Data Science. https://towardsdatascience.com/efficient-time-series-using-pythons-pmdarima-library-f6825407b7f0/

Learn Dapper. (17 de Octubre de 2024). https://www.learndapper.com/

Lucide. (s.f.). *Home.* https://lucide.dev/icons/

Martin, R.C. (2012). *The Clean Architecture*. The Clean Code Blog. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

MDN. (15 de abril de 2025). *Getting started with React - Learn web development | MDN*. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started

Meta. (17 de Mayo de 2025). *Quick start*. https://facebook.github.io/prophet/docs/quick_start.html

Microsoft. (13 de Abril de 2018). *Extracción, transformación y carga de datos (ETL)*. Microsoft Learn: https://learn.microsoft.com/es-es/azure/architecture/data-guide/relational-data/etl

Microsoft. (2 de Enero de 2025a). *¿Qué es SQL Server?* https://learn.microsoft.com/es-es/sql/sql-server/what-is-sql-server?view=sql-server-ver16

Microsoft. (23 de Abril de 2025b). *¿Qué es SQL Server Management Studio (SSMS)?*. https://learn.microsoft.com/es-es/ssms/sql-server-management-studio-ssms

Microsoft. (3 de Diciembre de 2022a). *C#.* Microsoft Dotnet. https://dotnet.microsoft.com/es-es/languages/csharp

Microsoft. (23 de Diciembre de 2022c). *Visual Studio*. https://visualstudio.microsoft.com/es/

Microsoft. (3 de Diciembre de 2022b). *¿Qué es .NET?* https://dotnet.microsoft.com/es-es/learn/dotnet/what-is-dotnet

Microsoft. (s.f.a). *Documentation - TypeScript for JavaScript programmers*. TypeScript: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Microsoft. (s.f.b). *What is Azure DevOps - Azure DevOps*. https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

Microsoft. (s.f.c). *Power BI: visualización de datos | Microsoft Power Platform*. https://www.microsoft.com/es-es/power-platform/products/power-bi

Muradas, Y. (3 de Junio de 2019). *Qué es Postman y primeros pasos*. OpenWebinars: https://openwebinars.net/blog/que-es-postman/

Bajaj, A. (18 de Agosto de 2023). *ARIMA & SARIMA: Pronóstico de series temporales en el mundo real*. Neptune.ai: https://neptune.ai/blog/arima-sarima-real-world-time-series-forecasting-guide

Numpy. (12 de Mayo de 2022). *¿Qué es NumPy?* https://numpy.org/devdocs/user/whatisnumpy.html

Oracle. (2 de Julio de 2022). *¿Qué es ETL?* Oracle Cloud Infrastructure (OCI). https://www.oracle.com/co/integration/what-is-etl/

Postman. (10 de Septiembre de 2019). *Your Complete API Platform, From Design to Delivery*. https://www.postman.com/

Predik Data-Driven. (2024). *Analítica de Datos Para Empresas: Guía Completa 2024*. https://predikdata.com/es/analitica-de-datos-5-ventajas-para-la-toma-de-decisiones

Prettier. (s.f.). *Why prettier? · Prettier*. https://prettier.io/docs/why-prettier

Probabilidad y Estadística. (2 de enero de 2024). *Modelo ARIMA*. https://www.probabilidadyestadistica.net/modelo-arima/

React Router. (s.f.). *Picking a mode*. https://reactrouter.com/start/modes

Refine. (11 de Septiembre de 2024). *A Complete guide to pnpm*. https://refine.dev/blog/how-to-use-pnpm/#what-is-pnpm

Saadeddin, Z. (10 de Septiembre de 2024). *ARIMA para la previsión de series temporales: Guía completa*. Datacamp. https://www.datacamp.com/es/tutorial/arima

Sánchez, A. (12 de Mayo de 2022). *La librería Numpy*. Aprende con Alf. https://aprendeconalf.es/docencia/python/manual/numpy/

Schwaber, K., & Sutherland, J. (2020). *La guía definitiva de Scrum: las reglas del juego.* https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf

Shadcn. (s.f.). *Introduction*. shadcn/ui: https://ui.shadcn.com/docs

Sharma, P. (4 de Abril de 2025). *How to Save and Load Machine Learning Models in Python Using Joblib Library?* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/02/how-to-save-and-load-machine-learning-models-in-python-using-joblib-library/

Smith, A., & Smith, T. (23 de Octubre de 2023). *pmdarima*. https://pypi.org/project/pmdarima/

Statsmodels. (3 de Octubre de 2024). *Home.* https://www.statsmodels.org/stable/index.html

SYDLE. (24 de Noviembre de 2023). *What is KPI (Key Performance Indicator)? Definition, Uses, and Examples*. https://www.sydle.com/blog/kpi-615de90225ce5d3ef29a5570

Tableau. (s.f.). *What is Tableau?* https://www.tableau.com/why-tableau/what-is-tableau

Taylor, S. J., & Letham, B. (2017). *Forecasting at scale*. PeerJ Preprints.

Testing Library. (15 de Septiembre de 2021). *Introduction*. https://testing-library.com/docs/

TRBL Services. (20 de Octubre de 2021). *C# (C Sharp): Qué es, dónde se utiliza y para qué sirve*. TRBL Services. https://trbl-services.eu/blog-c-sharp-que-es-para-que-sirve/

Universidad Nacional Autónoma de México [UNAM]. (12 de Febrero de 2013). *Modelación Arima.* http://www.ptolomeo.unam.mx:8080/jspui/bitstream/132.248.52.100/363/7/A7.pdf

Vinugayathri. (30 de Mayo de 2019). *Why Should You Use xUnit? A Unit Testing Framework For .NET*. Clarion Technologies. https://www.clariontech.com/blog/why-should-you-use-xunit-a-unit-testing-framework-for-.net

Vitest. (s.f.). *Why vitest*. https://vitest.dev/guide/why.html

VoidZero. (s.f.). *Introducción*. https://es.vite.dev/guide/

Xairó, A. (30 de Julio de 2023). *¿Para qué sirven los indicadores clave de rendimiento (KPI) ?* PayFit. https://payfit.com/es/contenido-practico/indicadores-clave-de-rendimiento/

xUnit.net. (29 de Marzo de 2019). *About xUnit.net*. https://xunit.ne