

**PROYECTO DE GRADO**

**“DISEÑO E IMPLEMENTACIÓN DEL BACK-END DE UN SISTEMA DE GESTIÓN DE INVENTARIOS PARA LA TIENDA DE AZULEJOS "SAN CLEMENTE" UBICADA EN VALLE ALTO DE LA PROVINCIA PUNATA DEL DEPARTAMENTO DE COCHABAMBA”**

**POSTULANTE: JAROD ZARIAN CADIZ SALANOVA**

**TUTOR: 6TO SEMESTRE Y ING. ALBINO MARTIN**

**Proyecto de Grado para optar al Grado Académico de Técnico**

**Superior en Sistemas Informáticos**

**COCHABAMBA- BOLIVIA**

**GESTIÓN 2025**

## RESUMEN

**TITULO: “DISEÑO E IMPLEMENTACIÓN DEL BACK-END DE UN SISTEMA DE GESTIÓN DE INVENTARIOS PARA LA TIENDA DE AZULEJOS "SAN CLEMENTE" UBICADA EN VALLE ALTO DE LA PROVINCIA PUNATA DEL DEPARTAMENTO DE COCHABAMBA”**

### AUTOR (ES): JAROD ZARIAN CADIZ SALANOVA

## PROBLEMATICA

La tienda “San Clemente” enfrenta dificultades en la gestión de su inventario debido al uso de registros manuales en hojas de cálculo. Esto ocasiona pérdida de información, duplicación de datos, imposibilidad de acceder a información en tiempo real y falta de reportes confiables para la toma de decisiones.

## OBJETIVO GENERAL

Diseñar e implementar el **backend de un sistema de gestión de inventarios** que optimice los procesos de control, almacenamiento y administración de datos en la tienda de azulejos San Clemente.

## CONTENIDO

El proyecto propone el desarrollo de un **sistema de gestión de inventario** para **automatizar procesos administrativos**, optimizando el control de información y la toma de decisiones. Se aplica la **metodología ágil Scrum**, usando **Python** por su versatilidad y **PostgreSQL** por su robustez. Además, se establecen **roles y una planificación estructurada** para asegurar un producto eficiente y alineado con las necesidades de los usuarios.

|  |  |  |
| --- | --- | --- |
| CARRERA |  | Sistemas Informáticos |
| PROFESOR GUIA |  | Ing. Albino Martin |
| PERIODO DE INVESTIGACIÓN |  | 4 meses |
| E – MAIL |  | cadizsalanovajarodzarian@gmail.com |
| CELULAR |  | 63870168 |

**DEDICATORIA**

Dedico este proyecto a mi fiel compañera de batallas, mi laptop DELL, que soportó largas horas de código, errores inesperados y bugs repentinos sin rendirse.

Al café, por ser el combustible que mantuvo despierto mi tormento en cada madrugada para poder acabar este tormento de programación.

Y a TikTok, porque entre tutoriales de programación, distracciones y risas, fue el recordatorio perfecto de que incluso en medio del trabajo, siempre hay espacio para una pausa y una sonrisa.

“Gracias a cada uno por acompañarme en este viaje hacia la culminación de mi proyecto de grado.”

**AGRADECIMIENTOS**

Quiero expresar mi más sincero agradecimiento a la tienda SAN CLEMENTE, por el apoyo brindado y por permitirme tomar como inspiración su entorno de trabajo para el desarrollo de este sistema de inventario. Su colaboración y disposición fueron fundamentales para comprender las necesidades reales del proyecto.

A mi tutor, Ing. Albino Martin, por su orientación y consejos durante el proceso. Su experiencia fue una guía que me permitió culminar con éxito este trabajo.

A todas las personas que, de una u otra manera, aportaron su tiempo, conocimiento y ánimo, les extiendo también mi más profundo agradecimiento.

**INDICE**

## CAPÍTULO I – MARCO REFERENCIAL

“DISEÑO E IMPLEMENTACIÓN DEL BACK-END DE UN SISTEMA DE GESTIÓN DE INVENTARIOS PARA LA TIENDA DE AZULEJOS "SAN CLEMENTE" UBICADA EN VALLE ALTO DE LA PROVINCIA PUNATA DEL DEPARTAMENTO DE COCHABAMBA”

### 1.1 INTRODUCCIÓN\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_pag.9

### 1.2 ANTECEDENTES

### 1.3 PLANTEAMIENTO Y FORMULACIÓN DEL PROBLEMA

**1.3.1 Descripción del Problema**

**1.3.2 Identificación del Problema**

**1.3.3 Formulación del Problema**

### 1.4 OBJETIVOS

**1.4.1 Objetivo General**

**1.4.2 Objetivos Específicos**

**1.5 JUSTIFICACION**

**1.5.1 Justificación social**

**1.5.2 Justificación técnica**

**1.5.3 Justificación económica (solo si corresponde)**

**1.6 ALCANCES**

**1.7 LÍMITES**

### 1.8 ENFOQUE METODOLÓGICO

**1.8.1 Metodología de Desarrollo**

**1.8.2 Técnicas de Recolección de Información**

**1.8.3 Población del Objeto de Estudio**

## CAPÍTULO II – MARCO TEORICO

### 2.1 INTRODUCCIÓN

### 2.2 SISTEMA DE INFORMACIÓN – APLICACIÓN WEB - INFRAESTRUCTURA

### 2.3 HERRAMIENTAS PARA EL DESARROLLO DE SOFTWARE

**2.3.1 Herramientas para la implementación del FRONT-END**

**2.3.1.1**

**2.3.1.2**

**2.3.2 Herramientas para la implementación del BACK-END**

**2.3.3 SGBD**

**2.3.4 Otra herramienta adicional (ejemplo control de versiones u otro)**

**La descripción de las herramientas dentro de este apartado puede realizarse en base a: HISTORIA, CARACTERISTICAS, VENTAJAS, DESVENTAJAS, TABLA DE EVOLUCION DE LA HERRAMIENTA (si corresponde), aproximadamente de unas 20 a 30 páginas. Ya que pueden incluir imágenes, tablas, etc.)**

### 2.4 METODOLOGIAS ÁGILES

**2.4.1 Definición de metodologías ágiles**

**2.4.2 Principios del Manifiesto ágil**

**2.4.3 Características de la metodología ágil**

**2.4.4 Metodología ágiles vs. Metodología tradicionales**

**2.5 METODOLOGIA SCRUM**

**2.5.1 Historia**

**2.5.2 Forma de trabajo**

**2.5.3 Ciclo de Vida**

**2.5.4 Roles de usuario**

**2.3.5 Características**

**2.3.6 Ventajas**

**2.3.7 Desventajas**

**(Los puntos dentro este apartado son solo referenciales ya que se puede ampliar o reducir en función a la metodología ágil elegida, aunque se recomienda utilizar SCRUM, aproximadamente de unas 10 a 15 páginas. Ya que deben incluir imágenes, tablas, etc.)**

**2.4 FUNDAMENTOS TEORICOS DEL SISTEMA (depende del sistema)**

**2.4.1 Definición de conceptos**

**2.4.2 Terminología del área de inventario (es solo un ejemplo)**

**2.4.3 Manejo de información**

**Los puntos a describir dentro de este apartado son solo referenciales ya que en función al tipo de sistema a desarrollar se especificarán los términos y la información que corresponda, aproximadamente de unas 5 a 10 páginas. Ya que pueden incluir imágenes, tablas, etc.)**

## CAPÍTULO III – MARCO APLICATIVO O PROPUESTA DE SOLUCIÓN

### 3.1 INTRODUCCIÓN

### 3.2 REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

**3.2.1 Requerimientos Funcionales**

**3.2.2 Requerimientos no funcionales**

**3.3 APLICACIÓN DE LA METODOLOGIA SCRUM**

**3.3.1 Historias de usuario**

**3.3.2 Product Backlog**

**3.3.3 Sprint Backlog**

**3.3.4 Reuniones**

**3.3.5 Roles de usuario**

**3.3.6 Planificación de Sprints**

**3.3.6.1 Primer Sprint**

**3.3.6.2 Segundo Sprint**

**3.3.6.3 Tercer Sprint**

**3.3.6.4 Cuarto Sprint**

**La descripción de este apartado se desarrollará en función a la metodología ágil elegida, aproximadamente de unas 20 a 50 páginas. Ya que deben incluir imágenes, tablas, gráficos, diagramas, etc. Cada uno de los diagramas o tablas deben estar detalladamente descritos.**

**3.4 DISEÑO DE LA BASE DE DATOS**

**3.4.1 Descripción del tipo de diagrama a utilizar (sustentar con bibliografía de investigación)**

**3.4.2 Simbología – Elementos – etc.**

**3.4.3 Diagrama ER del proyecto**

**3.4.4 Descripción de cada una de las tablas de la BD**

**3.4.5 Diccionario de Datos, ETC.**

**3.5 FASE DE CONSTRUCCIÓN**

**3.5.1. Herramientas de Implementación**

**3.5.2 Herramientas de Diseño**

En función a la metodología ágil elegida desarrollar cada uno de los sprints ó módulos del sistema

**3.5.3 Entorno de Desarrollo**

**3.5.3.1 Implementación del Acceso del Sistema**

**3.5.3.2 Implementación del módulo de usuario**

**3.5.3.3**

**3.5.3.4**

**3.5.4 Diagrama de Actividades Administración de Usuarios**

**3.5.5 Fase de Pruebas**

**3.5.6 Requerimientos del sistema**

**3.5.6.1 Requerimientos de Hardware**

**3.5.6.2 Requerimientos de Software**

**3.5.7 Instalación del sistema**

**3.5.7.1 Instalación de los Programas**

**3.5.7.2 Instalación de los archivos**

**3.5.7.3 Utilización del Sistema**

**CAPÍTULO IV – CONCLUSIONES Y RECOMENDACIONES**

**4.1 CONCLUSIONES**

**4.1.1 Conclusiones acerca de los objetivos**

**4.1.2 Conclusiones acerca de las herramientas**

**4.1.3 Conclusiones acerca de la metodología**

**4.2 RECOMENDACIONES**

**4.2.1 Recomendaciones acerca de la instalación**

**4.2.2 Recomendaciones acerca del uso del sistema**

**4.2.3 Recomendaciones acerca de las actualizaciones**

**BIBLIOGRAFIA**

**Índice de figuras**

Figura 1: Modelo CVS........................................................................................................16

Figura 2: Ciclo de vida Struts.............................................................................................18

Figura 3: Ciclo de vida JSF..................................................................................................20

Figura 4: Ciclo de vida WebWork ......................................................................................21

Figura 5: Ciclo de vida Tapestry ........................................................................................23

Figura 6: Secuencia de request Struts...............................................................................29

**Índice de tablas**

Tabla 1: Comparación metodologías agiles vs metodologías tradicionales....................................26

Tabla 2: Elaboración de Product Backlog .......................................................................................38

Tabla 3: Ciclo de vida Iterativo-Incremental…………………………………………........................................61

## CAPÍTULO 1

## MARCO REFERENCIAL

* 1. **INTRODUCCIÓN**

La transformación digital ha impulsado a las empresas a modernizar sus procesos internos, optimizando recursos mediante sistemas robustos y automatizados. En el ámbito comercial, la gestión del inventario constituye un pilar esencial para garantizar la disponibilidad de productos, reducir errores y mantener la competitividad.

Las tiendas de azulejos enfrentan una complejidad particular en la administración de stock, debido a la variedad de modelos, colores, tamaños y lotes, lo cual requiere un sistema confiable y escalable. En este contexto, el **backend de un sistema de gestión de inventarios** constituye el núcleo de la solución tecnológica, ya que concentra la lógica de negocio, la seguridad de los datos y la comunicación con distintas interfaces cliente (aplicaciones web, móviles u otros sistemas).

El presente proyecto se centra en el **diseño e implementación de la capa de backend**, que permitirá registrar y procesar las operaciones de inventario (entradas, salidas, movimientos), gestionar usuarios y roles, generar reportes dinámicos, emitir notificaciones automáticas y garantizar el acceso a información en tiempo real mediante una API segura y escalable.

Con esta propuesta se busca proporcionar a la tienda San Clemente una infraestructura tecnológica sólida y preparada para integrarse con diferentes frontends en el futuro, optimizando la administración del negocio y asegurando su crecimiento sostenible.

**1.2 ANTECEDENTES**

**a) Misión**

Brindar a los clientes materiales de revestimiento y construcción de alta calidad, garantizando un servicio eficiente, personalizado y confiable.

**b) Visión**

Consolidarse como la tienda líder en el rubro de azulejos y revestimientos en la región, apoyándose en la innovación tecnológica para optimizar procesos y ofrecer una mejor experiencia al cliente.

**c) Descripción organizacional de la empresa**

La tienda **San Clemente**, fundada en 2018, es una empresa familiar dedicada a la comercialización de azulejos y revestimientos cerámicos. Ubicada estratégicamente en el centro de la ciudad, cuenta con un almacén y una sala de exposición de productos. Actualmente, su control de inventario se realiza de forma manual mediante hojas de cálculo, lo cual genera errores frecuentes, falta de trazabilidad y dificultades para responder de manera ágil a la demanda del mercado.

**1.3 JUSTIFICACIÓN**

1. **Justificación técnica**

El desarrollo de un **backend robusto** permitirá implementar una base de datos relacional y una API RESTful para gestionar entradas, salidas, movimientos de inventario y usuarios. Esto centralizará la información, garantizará la integridad de los datos y permitirá la escalabilidad futura del sistema.

1. **Justificación económica**

La automatización de procesos reducirá pérdidas por errores humanos, optimizará la reposición de productos y disminuirá costos asociados a fallos en la gestión del stock. A mediano plazo, la inversión en el backend generará un retorno a través de ahorro de recursos y mayor competitividad.

1. **Justificación social**

La solución contribuirá a mejorar la organización interna, reduciendo la carga de trabajo del personal y aumentando la eficiencia en la atención al cliente. Asimismo, permitirá una administración más transparente y confiable del inventario.

1. **Justificación medioambiental (opcional, si corresponde de acuerdo al proyecto)**

Aunque no es el eje principal del proyecto, al optimizar el inventario y reducir errores en los pedidos, se disminuirá el desperdicio de productos y materiales, contribuyendo indirectamente a un mejor aprovechamiento de los recursos.

**1.4 PLANTEAMIENTO Y FORMULACIÓN DEL PROBLEMA**

**1.4.1 Descripción del problema**

La tienda “San Clemente” enfrenta dificultades en la gestión de su inventario debido al uso de registros manuales en hojas de cálculo. Esto ocasiona pérdida de información, duplicación de datos, imposibilidad de acceder a información en tiempo real y falta de reportes confiables para la toma de decisiones.

**1.4.2 Formulación del Problema**

¿La implementación de un backend para un sistema de gestión de inventarios permitirá mejorar la precisión, seguridad y eficiencia en el manejo de stock en la tienda de azulejos San Clemente?

**1.5 OBJETIVOS**

**1.5.1 Objetivo General**

Diseñar e implementar el **backend de un sistema de gestión de inventarios** que optimice los procesos de control, almacenamiento y administración de datos en la tienda de azulejos San Clemente.

**1.5.2 Objetivos Específicos**

1. Analizar los procesos de la tienda para identificar los requerimientos del backend.
2. Diseñar una base de datos relacional que administre productos, proveedores, usuarios y transacciones.
3. Implementar una API RESTful que permita la comunicación segura entre el backend y futuros frontends.
4. Incorporar mecanismos de seguridad como autenticación, autorización y auditoría de operaciones.
5. Realizar pruebas de rendimiento, integración y seguridad para garantizar la estabilidad del backend.

**1.6 ESTUDIO DE FACTIBILIDAD**

El estudio de factibilidad evalúa si el proyecto es viable desde diferentes aspectos.

1. **Factibilidad social**

El proyecto impactará positivamente al personal al reducir la carga de trabajo manual y mejorar la precisión de los registros. Los clientes se verán beneficiados con una atención más rápida y confiable.

1. **Factibilidad técnica**

El backend será desarrollado con tecnologías ampliamente utilizadas (por ejemplo: Node.js/Java/Kotlin para servicios, y MySQL/PostgreSQL como gestor de base de datos). El sistema será escalable, integrable y accesible desde distintos dispositivos.

1. **Factibilidad económica**

La inversión inicial se destina al desarrollo del backend y su despliegue en un servidor. A largo plazo, se recuperará mediante la reducción de errores, el ahorro de tiempo operativo y la mejora en la rotación del inventario.

**1.7 ALCANCES**

* Gestión de inventario: entradas, salidas y movimientos internos.
* API RESTful disponible para ser consumida por aplicaciones web o móviles.
* Generación de reportes dinámicos en formatos digitales (JSON, CSV, PDF).
* Gestión de usuarios con diferentes roles y permisos.
* Notificaciones automáticas de stock bajo.

**1.8 LÍMITES**

* No incluirá en esta fase un frontend visual.
* No tendrá integración inmediata con sistemas externos (ERP o contabilidad).
* No se implementará comercio electrónico en la primera versión.

**1.9 PLANIFICACIÓN Y CONTROL**

El proyecto será desarrollado bajo metodología ágil con planificación iterativa en **sprints.** Se utilizarán herramientas de control de versiones (Git) y tableros de gestión de tareas (Trello, Jira) para monitorear avances, entregables y riesgos.

**1.10 ENFOQUE METODOLÓGICO**

**1.10.1 Metodología de Desarrollo**

Se adoptará **Scrum**, priorizando la entrega incremental de funcionalidades del backend: diseño de base de datos, creación de API, seguridad y pruebas.

**1.10.2 Técnicas de Recolección de Información**

* Entrevistas con los encargados de la tienda.
* Observación directa de procesos actuales.
* Revisión documental de registros y hojas de cálculo.

**1.10.3 Población del Objeto de Estudio**

La población está compuesta por el personal administrativo de la tienda San Clemente encargado de la gestión del inventario, así como los responsables de la toma de decisiones relacionadas con el stock.

**CAPÍTULO 2**

**MARCO TEÓRICO**

**2.1 INTRODUCCIÓN**

El desarrollo de sistemas de información modernos requiere no solo habilidades técnicas especializadas, sino también un marco teórico sólido que permita comprender los elementos fundamentales sobre los cuales se construyen las soluciones. El marco teórico constituye la base conceptual que orienta la planificación, el diseño y la implementación de sistemas de software.

En este contexto, el backend constituye el núcleo lógico de cualquier aplicación, ya que se encarga de procesar la información, aplicar reglas de negocio, almacenar y gestionar datos de forma segura y entregar resultados al frontend. Además, facilita la interoperabilidad con otros sistemas mediante APIs, implementa mecanismos de autenticación y autorización, y asegura la coherencia e integridad de la información.

En el Sistema de Gestión de Inventarios para la tienda de azulejos San Clemente, el backend es especialmente crítico, ya que maneja información sensible de inventarios, ventas, clientes y proveedores. Su diseño eficiente no solo optimiza los procesos internos, sino que también mejora la experiencia del usuario y permite a la organización adaptarse a cambios en la demanda o volumen de transacciones.

Este capítulo explora los conceptos, herramientas y metodologías del backend, incluyendo su evolución histórica, lenguajes de programación, frameworks, bases de datos, seguridad, control de versiones y despliegue, así como la metodología de desarrollo adoptada. Esta revisión teórica ofrece un marco integral que sustenta la implementación del sistema (Pressman, 2014; Sommerville, 2011).

**2.2 HERRAMIENTAS BACKEND**

El backend, o “capa de servidor”, es el componente encargado de recibir las solicitudes del frontend, procesarlas de acuerdo con la lógica de negocio, interactuar con la base de datos y devolver resultados al usuario. Su correcta implementación asegura seguridad, escalabilidad, integridad de datos y eficiencia operativa.

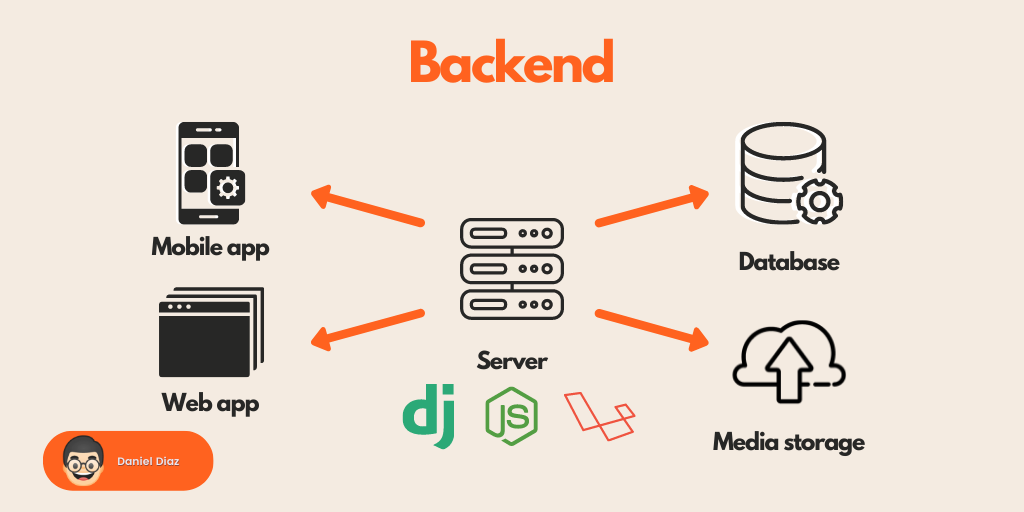


FIGURA 1. Backend de un sistema (URL: <https://www.initiumsoft.com/blog_initium/back-end-que-es/> )

Entre sus funciones destacan:

* Procesamiento de información y cálculos complejos: desde sumar inventarios hasta generar reportes financieros.
* Gestión de la lógica de negocio: aplica políticas internas de la empresa, como alertas de stock bajo o reglas de facturación.
* Seguridad: controla la autenticación de usuarios, cifrado de información, gestión de roles y auditoría de eventos críticos.
* Interoperabilidad: mediante APIs, permite que otros sistemas o aplicaciones móviles accedan a información de forma estandarizada y segura.

Una arquitectura backend bien diseñada también garantiza escalabilidad horizontal, permitiendo manejar un aumento en usuarios y transacciones sin degradar el rendimiento del sistema.

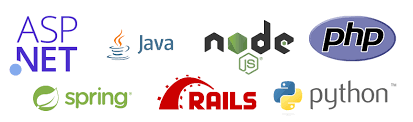


FIGURA 2. Tipos de lenguaje(URL: <https://share.google/images/VrUf3xna9nOdUpEgg> )

**2.2.1 HISTORIA Y EVOLUCIÓN DEL BACKEND**

El backend ha evolucionado de manera significativa a medida que las tecnologías y necesidades de negocio se han transformado:

1. Década de 1960 – Mainframes: Sistemas monolíticos centralizados. Todo el procesamiento ocurría en un único servidor, y los usuarios accedían mediante terminales “tontas”. Este modelo era robusto y seguro, pero difícil de escalar.
2. Décadas de 1980-1990 – Arquitectura Cliente-Servidor: La carga se distribuyó entre cliente y servidor. Los clientes procesaban parte de la información, mientras los servidores gestionaban la lógica de negocio y los datos. Esto permitió mayor interactividad, aunque aumentó la complejidad de mantenimiento y compatibilidad.
3. Década de 2000 – Aplicaciones Web y APIs: Con la expansión de Internet, las aplicaciones adoptaron una arquitectura de tres capas: presentación, lógica de negocio y datos. Las APIs REST y SOAP facilitaron la comunicación entre sistemas heterogéneos y la integración de servicios externos.
4. Actualidad – Microservicios y Cloud: Los sistemas modernos adoptan arquitecturas de microservicios, donde cada servicio es independiente, escalable y desplegable de manera autónoma. La computación en la nube (AWS, Google Cloud, Azure) permite alta disponibilidad, escalabilidad automática y redundancia de datos. La seguridad se ha reforzado con cifrado en tránsito y reposo, autenticación multifactor y control granular de roles (Stonebraker et al., 1986; Schwaber & Sutherland, 2020).

**2.2.2 LENGUAJE DE PROGRAMACIÓN: PYTHON**

Python, creado por Guido van Rossum en 1991, es un lenguaje interpretado, multiparadigma y de alto nivel. Se diseñó para ser legible y simple, reduciendo la curva de aprendizaje y facilitando la colaboración en equipos de desarrollo.

* + **EVOLUCIÓN:**
* Python 2 (2000): Popularizó el lenguaje pero tenía limitaciones con Unicode.
* Python 3 (2008): Introdujo mejoras en legibilidad, Unicode completo y compatibilidad con librerías modernas.

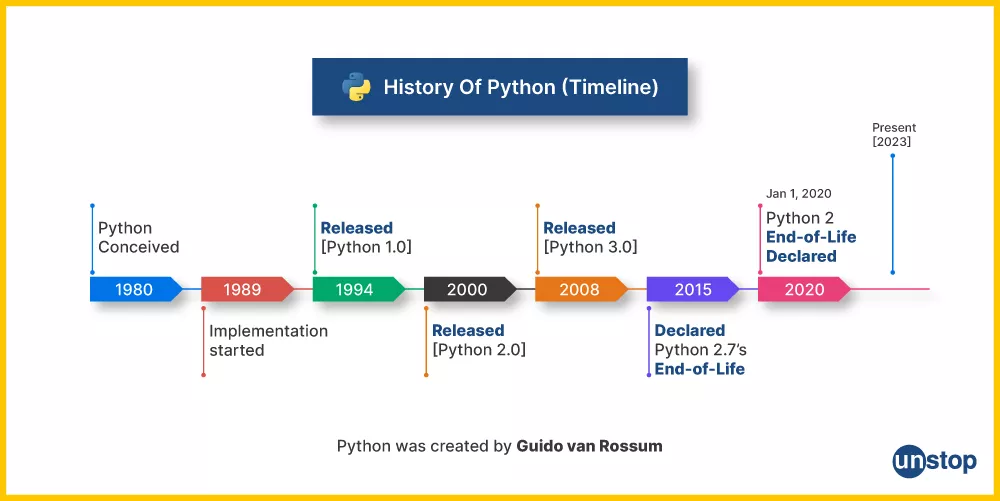


FIGURA 3. Historia de python(URL: [**https://unstop.com/blog/what-is-python**](https://unstop.com/blog/what-is-python) )

* **VENTAJAS:**
* Sintaxis clara y legible.
* Ecosistema amplio de librerías y frameworks (Flask, FastAPI, Django, Pandas, NumPy).
* Multiparadigma: soporta POO, funcional e imperativo.
* Portabilidad en distintos sistemas operativos.
* **DESVENTAJAS:**
* Rendimiento menor comparado con lenguajes compilados.
* Dependencia de librerías externas para funcionalidades avanzadas.

1. **HISTORIA DE PYTHON**

Python es un lenguaje de programación de alto nivel, interpretado y multiparadigma, que se ha convertido en uno de los más influyentes en el ámbito académico, científico e industrial. Su creación se remonta a finales de la década de 1980, cuando Guido van Rossum, un programador holandés del *Centrum Wiskunde & Informatica* (CWI) en Ámsterdam, comenzó a desarrollar un nuevo lenguaje que combinara la simplicidad del lenguaje de scripting con la potencia de lenguajes estructurados como C (Van Rossum, 1991).

El proyecto se inició en diciembre de 1989, durante el período vacacional de Van Rossum, inspirado en el lenguaje ABC, también desarrollado en el CWI. Aunque ABC tenía un diseño pedagógico atractivo, presentaba limitaciones en aspectos de extensibilidad y manejo de errores. Van Rossum buscó crear un lenguaje más flexible y práctico, que resultara intuitivo tanto para principiantes como para expertos, y que a la vez fuera compatible con sistemas UNIX (Van Rossum & Drake, 2009).

La primera versión oficial, Python 0.9.0, fue liberada en febrero de 1991, e incluía características que aún hoy distinguen al lenguaje: manejo de excepciones, funciones y módulos. En ese mismo año se publicó la versión 1.0, marcando el inicio de su adopción en la comunidad académica y de software libre (Lutz, 2013).

Durante la década de 1990, Python ganó popularidad debido a su sintaxis clara y legible, basada en la indentación en lugar de llaves o palabras clave. En 2000, la administración del lenguaje pasó a la recién creada Python Software Foundation (PSF), lo que consolidó su carácter comunitario y abierto (Python Software Foundation, 2001).

Ese mismo año, se liberó Python 2.0, que introdujo características significativas como la recolección de basura mediante conteo de referencias y *list comprehensions*. Sin embargo, la evolución del lenguaje llevó a una transición mayor con Python 3.0, lanzado en 2008, que no fue retrocompatible con versiones anteriores. Python 3 modernizó la sintaxis, mejoró el manejo de cadenas de texto (Unicode por defecto) y eliminó redundancias del lenguaje, aunque su adopción global fue lenta y coexistió con Python 2 hasta 2020, cuando este último dejó de recibir soporte oficial (Van Rossum, 2010).

En la actualidad, Python es uno de los lenguajes más utilizados en el mundo, especialmente en áreas como inteligencia artificial, ciencia de datos, desarrollo web y automatización. Según el índice TIOBE (2023), Python ha ocupado en varias ocasiones el primer lugar como el lenguaje de programación más popular, lo que confirma el impacto global de su diseño sencillo, su comunidad activa y la riqueza de sus bibliotecas.

**2.2.3 FRAMEWORK DE DESARROLLO: FLASK Y FASTAPI**

## **FLASK (2010)**

Flask es un **microframework minimalista y flexible** escrito en Python. Fue lanzado en 2010 y está basado en dos librerías fundamentales: **Werkzeug** (para el enrutamiento y servidor WSGI) y **Jinja2** (para plantillas HTML). Su filosofía es ofrecer solo lo esencial, dejando al desarrollador la libertad de añadir librerías adicionales según las necesidades del proyecto.

Entre sus características principales se destacan:

* **Simplicidad y rapidez de aprendizaje**, lo que lo hace ideal para prototipos, proyectos pequeños o medianos, así como entornos educativos.
* Amplia comunidad y ecosistema de extensiones como **Flask-Login** (gestión de usuarios) o **Flask-SQLAlchemy** (bases de datos).
* Gran uso en la exposición de modelos de **Machine Learning** mediante servicios web sencillos.
* **VENTAJAS:**
* Fácil de usar y aprender.
* Comunidad madura con gran cantidad de recursos.
* Alta flexibilidad y personalización.
* **LIMITACIONES:**
* No incluye soporte nativo para asincronía.
* Al crecer la aplicación puede volverse difícil de estructurar.
* Su rendimiento es bueno, pero menor en comparación con frameworks modernos como FastAPI.

## **FASTAPI (2018)**

FastAPI es un framework moderno lanzado en 2018, diseñado específicamente para construir **APIs de alto rendimiento**. Se basa en **Starlette** (para la capa web) y **Pydantic** (para validación de datos), lo que lo hace rápido, seguro y eficiente.

Entre sus características principales se incluyen:

* **Soporte nativo de asincronía** con async/await, lo que permite manejar múltiples solicitudes concurrentes eficientemente.
* **Documentación automática** en **Swagger UI** y **ReDoc**, generada directamente a partir del código.
* Validación de datos y serialización automática gracias al tipado de Python.
* Excelente rendimiento, comparable con Node.js y Go.
* Muy usado en **arquitecturas modernas de microservicios** y en **despliegues de modelos de Machine Learning e IA** en producción.
* **VENTAJAS:**
* Rendimiento superior.
* Validación y documentación automática.
* Uso del tipado de Python para mayor legibilidad y autocompletado en IDEs.
* Ideal para proyectos escalables y APIs modernas.
* **LIMITACIONES:**
* Comunidad más joven (aunque en rápido crecimiento).
* Curva de aprendizaje algo más pronunciada si no se domina la asincronía ni los type hints.
* **COMPARATIVA TÉCNICA:**

|  |  |  |
| --- | --- | --- |
| Característica | Flask | FastAPI |
| Facilidad de aprendizaje | Muy alta | Media |
| Rendimiento | Bueno | Muy alto |
| Validación de datos | Manual | Automática (Pydantic) |
| Documentación automática | No nativa | Sí (Swagger/OpenAPI) |
| Escalabilidad | Media | Alta |
| Comunidad | Amplia | Creciente |

TABLA 1. Comparativa de flask y fastapi (Elaboración propia)

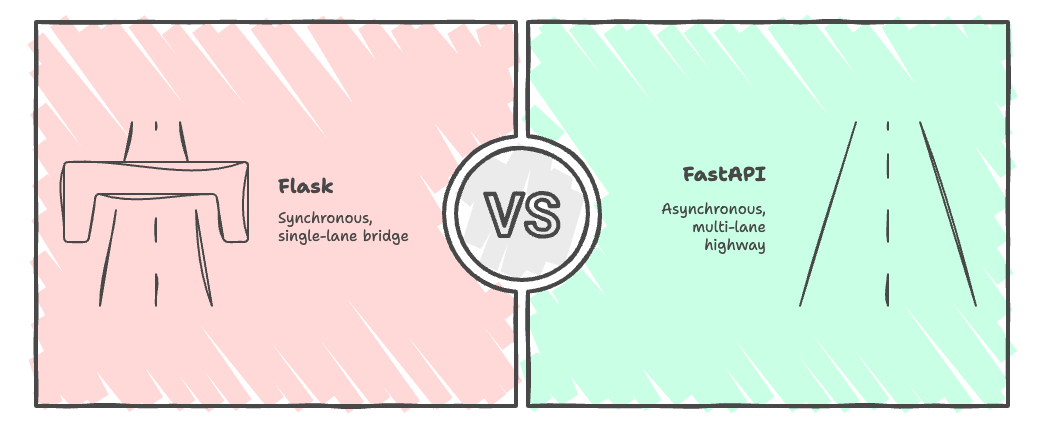
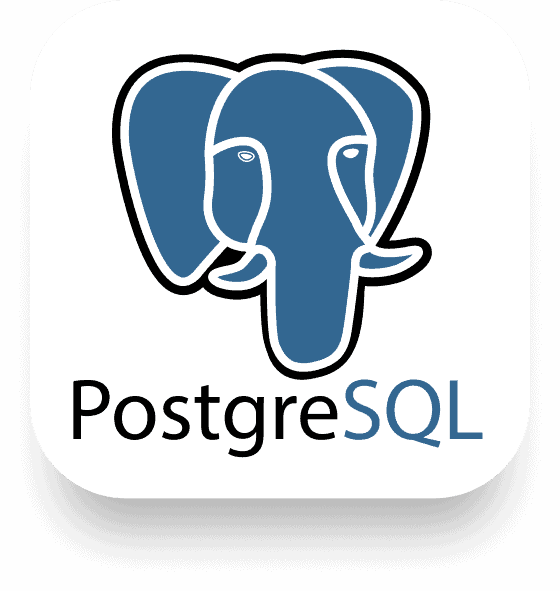


FIGURA 4. Como es el flask y fastapi (URL: <https://blog.gopenai.com/flask-vs-fastapi-why-flask-sucks-in-production-31ae52dd62e3> )

**2.2.4 BASE DE DATOS: POSTGRESQL**

* **HISTORIA:** PostgreSQL es un sistema de gestión de bases de datos relacional y objeto-relacional (ORDBMS) de código abierto, considerado uno de los más avanzados y robustos en la actualidad. Su historia se remonta a mediados de la década de 1980 en la Universidad de California, Berkeley, como parte de la evolución del proyecto Ingres, uno de los primeros sistemas de bases de datos relacionales (Stonebraker & Rowe, 1986).

El desarrollo inicial de PostgreSQL comenzó en 1986, liderado por el profesor Michael Stonebraker, bajo el nombre de POSTGRES. El objetivo de este proyecto era superar las limitaciones de Ingres, introduciendo características avanzadas como la extensibilidad, el manejo de objetos complejos y la integración con nuevas aplicaciones emergentes, incluyendo multimedia y datos geoespaciales (Stonebraker et al., 1991).

En 1989 se liberó la primera versión del sistema POSTGRES, y en 1994 se añadió el soporte para el lenguaje de consulta SQL, lo que marcó un hito clave en su adopción. En ese momento, el sistema pasó a denominarse Postgres95, como reflejo de su evolución hacia un estándar más cercano al mercado de bases de datos comerciales.

Posteriormente, en 1996, el proyecto fue renombrado oficialmente como PostgreSQL, combinando su herencia de *Postgres* con la incorporación plena de SQL como lenguaje principal. Desde entonces, el sistema ha evolucionado continuamente gracias a una activa comunidad de desarrolladores y a su modelo de software libre, garantizado bajo la licencia PostgreSQL, similar en espíritu a la licencia BSD (PostgreSQL Global Development Group, 2023).

Durante las décadas de 2000 y 2010, PostgreSQL se consolidó como un competidor sólido frente a sistemas comerciales como Oracle, Microsoft SQL Server y IBM Db2, gracias a su estabilidad, seguridad y conformidad con estándares. Además, introdujo características innovadoras como soporte nativo para JSON, XML, procedimientos almacenados en múltiples lenguajes y replicación lógica, lo que lo convirtió en una opción versátil para aplicaciones modernas (Momjian, 2015).

En la actualidad, PostgreSQL es utilizado por grandes empresas tecnológicas, gobiernos y organizaciones académicas. Es reconocido por su capacidad de manejar grandes volúmenes de datos, su alto grado de personalización y su rol pionero en integrar el paradigma relacional con el orientado a objetos. Su comunidad global de desarrolladores continúa publicando nuevas versiones de manera regular, asegurando mejoras en rendimiento, escalabilidad y soporte a nuevas arquitecturas de software.

* **VENTAJAS:**
* Cumple transacciones ACID.
* Extensible mediante módulos (PostGIS, PL/Python).
* Soporte para tipos de datos avanzados (JSONB, arrays, UUID).
* **EJEMPLO DE CONSULTA SQL:**

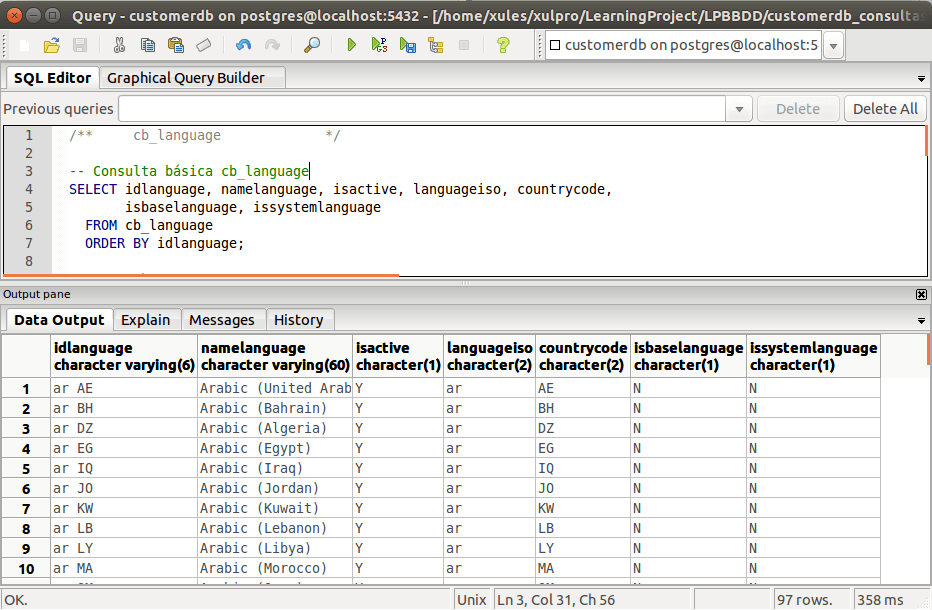


FIGURA 5. Ejemplo básico de una consulta sql(URL: <https://codigoxules.org/tutorial-postgresql-3-poblacion-de-datos-y-consultas-basicas/> )

* **COMPARATIVA:**

En esta tabla comparativa se mostrara las características que tiene postgresql con las demás bases de datos.

| **Característica** | **PostgreSQL** | **MySQL** | **MongoDB** |
| --- | --- | --- | --- |
| Tipo | Relacional | Relacional | NoSQL |
| Integridad de datos | Alta | Media | Media |
| Escalabilidad | Alta | Media | Muy alta |
| Consultas complejas | Excelente | Limitadas | Limitadas |

Tabla 2. Comparación de SGBD (Elaboración propia)

**2.2.5 SEGURIDAD Y AUTENTICACIÓN**

| **Parámetro** | **Herramienta/Protocolo** | **Función/Justificación** |
| --- | --- | --- |
| Autenticación | JWT | Permite autenticación sin estado, ideal para microservicios |
| Autorización | RBAC | Control de acceso por roles (administrador, empleado, cliente) |
| Contraseñas | Bcrypt | Hash seguro con sal para prevenir ataques |
| Auditoría | Logs | Registro de acciones críticas para trazabilidad y cumplimiento |

Tabla 3. Comparación de la seguridad (Elaboración propia)

**2.2.6 CONTROL DE VERSIONES Y DESPLIEGUE**

* Git: Control de versiones distribuido,7 facilita colaboración y seguimiento de cambios.
* GitHub/GitLab: Gestión de ramas, pull requests, CI/CD, revisión de código.
* Despliegue en la nube: AWS, Heroku, Railway, que permiten escalabilidad automática, monitoreo y despliegues confiables.

## 2.2.6.1 HISTORIA DE GITHUB

**GitHub** es una plataforma de colaboración y gestión de proyectos que se ha consolidado como una de las herramientas más influyentes en el desarrollo de software a nivel mundial. Su historia está estrechamente ligada a la evolución del sistema de control de versiones **Git**, creado por **Linus Torvalds** en 2005 para gestionar el código fuente del kernel de Linux (Loeliger & McCullough, 2012).

En sus inicios, Git fue concebido como una alternativa rápida y eficiente frente a sistemas centralizados como CVS o Subversion. Sin embargo, su complejidad técnica representaba una barrera para muchos desarrolladores. Esta limitación dio lugar a la creación de plataformas que ofrecieran una interfaz más amigable y servicios adicionales basados en Git, siendo GitHub la más exitosa.

GitHub fue fundado en **2008** por **Tom Preston-Werner, Chris Wanstrath, P. J. Hyett y Scott Chacon**. Su principal objetivo era ofrecer un entorno en la nube donde los desarrolladores pudieran alojar repositorios Git, colaborar en proyectos y compartir código de manera sencilla (Fogel, 2017). Lo que lo distinguió de otras soluciones fue la integración de características sociales como los forks, pull requests y issues, que fomentaron una cultura de trabajo colaborativo y abierto.

El crecimiento de GitHub fue exponencial. En tan solo un año, en **2009,** ya había superado a SourceForge y Google Code en popularidad. Para **2012**, contaba con más de un millón de usuarios y había alojado alrededor de tres millones de repositorios (Dabbish et al., 2012). Estas cifras lo consolidaron como la plataforma preferida por comunidades de software libre y también por empresas privadas que adoptaron la herramienta en sus flujos de trabajo.

En **2018**, GitHub alcanzó un nuevo hito cuando fue adquirida por **Microsoft** por 7.5 mil millones de dólares, lo que reflejó la importancia estratégica de la plataforma dentro del ecosistema global de desarrollo de software (Microsoft, 2018). Tras la adquisición, GitHub mantuvo su autonomía operativa, pero integró nuevas funcionalidades, como **GitHub Actions** (2019), que introdujo capacidades de integración y despliegue continuo (CI/CD), y **GitHub Copilot** (2021), un asistente de programación basado en inteligencia artificial desarrollado junto a OpenAI.

Actualmente, GitHub es la plataforma de desarrollo colaborativo más utilizada en el mundo, con más de 100 millones de usuarios (GitHub, 2023). No solo es un repositorio de código, sino un ecosistema completo que fomenta la innovación abierta, la cultura de colaboración global y el avance de proyectos en áreas tan diversas como la inteligencia artificial, la ciencia de datos y la educación.

* 1. **Metodología de Desarrollo: Scrum**

**2.3.1 HISTORIA DE SCRUM:**

El marco de trabajo Scrum surge como resultado de la necesidad de nuevas metodologías para gestionar proyectos complejos, especialmente en el ámbito del desarrollo de software, donde los modelos tradicionales, como el ciclo en cascada, resultaban insuficientes frente a la incertidumbre y el dinamismo del mercado (Pressman, 2010).

El término fue utilizado por primera vez en 1986 por Hirotaka Takeuchi e Ikujiro Nonaka en su artículo *The New New Product Development Game*, publicado en la revista *Harvard Business Review*. En dicho trabajo, los autores plantearon un enfoque innovador para el desarrollo de productos, inspirado en la metáfora del rugby: en lugar de avanzar de manera lineal y secuencial, los equipos debían trabajar de forma conjunta, flexible y multidisciplinaria, como en una “melé” o *scrum* en el rugby (Takeuchi & Nonaka, 1986).

Posteriormente, en la década de 1990, Ken Schwaber y Jeff Sutherland adaptaron y consolidaron estas ideas al desarrollo de software. En 1993, Sutherland aplicó por primera vez Scrum en Easel Corporation, y junto con Schwaber, presentaron formalmente el marco en la conferencia OOPSLA 1995 (Object-Oriented Programming, Systems, Languages & Applications), donde describieron su funcionamiento y beneficios frente a métodos tradicionales (Schwaber & Sutherland, 1995).

El punto de inflexión llegó en 2001, cuando Schwaber y Sutherland participaron en la redacción del Manifiesto Ágil, un documento histórico que sentó las bases de las metodologías ágiles y que estableció valores fundamentales como la colaboración con el cliente, la adaptabilidad al cambio y la entrega continua de valor (Beck et al., 2001). Desde entonces, Scrum se consolidó como el marco ágil más difundido en el mundo.

En 2010, Schwaber y Sutherland publicaron la primera versión de la Guía de Scrum (Scrum Guide), un documento oficial y de acceso abierto que describe los roles, artefactos y eventos que conforman el marco. Esta guía se ha actualizado periódicamente (2013, 2017 y 2020) para reflejar la evolución de la práctica y hacerla más universal, simplificada y aplicable a diferentes ámbitos más allá del software, como la gestión empresarial, la educación y el marketing (Schwaber & Sutherland, 2020).

En la actualidad, Scrum no solo es considerado una herramienta para el desarrollo ágil de software, sino un marco de trabajo versátil y ampliamente adoptado para la gestión de proyectos complejos en diversos sectores.



FIGURA 6. Dibujo del scrum(URL: <https://www.doonamis.com/metodologias-agiles-en-el-desarrollo-de-apps/> )

## PRINCIPIOS DE SCRUM

El marco de trabajo **Scrum** se sustenta en un conjunto de principios que guían su aplicación y lo diferencian de los modelos de gestión tradicionales. Estos principios garantizan que el trabajo en equipo sea iterativo, adaptable y orientado al valor entregado al cliente.

### **CONTROL EMPÍRICO DE PROCESOS**

Scrum se fundamenta en el empirismo, es decir, en la idea de que el conocimiento proviene de la experiencia y de la toma de decisiones basadas en lo observable. Por ello, el marco se rige por tres pilares esenciales:

* **Transparencia**: todos los aspectos significativos del proceso deben ser visibles para los miembros del equipo y los interesados.
* **Inspección**: los procesos y avances deben evaluarse regularmente.
* **Adaptación:** cuando se detecten desviaciones o problemas, el equipo debe ajustar el proceso de inmediato (Schwaber & Sutherland, 2020).

### **AUTOORGANIZACIÓN**

Scrum promueve equipos multidisciplinarios que se autoorganizan, lo que significa que los miembros deciden cómo llevar a cabo el trabajo sin depender de órdenes jerárquicas. Esto fomenta la autonomía, la creatividad y el compromiso (Takeuchi & Nonaka, 1986).

### **COLABORACIÓN**

La colaboración se entiende como el trabajo conjunto entre el equipo de desarrollo, el Product Owner y el Scrum Master. El marco enfatiza la comunicación constante, la negociación y la confianza mutua para lograr un producto de valor.

### **PRIORIZACIÓN BASADA EN EL VALOR**

Scrum asegura que el equipo siempre trabaje en lo más importante para el negocio o el cliente. El Product Owner es responsable de priorizar el **Product Backlog,** de modo que los entregables generen el máximo valor posible en cada iteración (Cohn, 2004).

### **TIMEBOXING (GESTIÓN DEL TIEMPO)**

Scrum establece límites temporales estrictos para sus eventos (Sprints, Daily Scrum, Sprint Review y Retrospective). El uso de “cajas de tiempo” permite enfocar al equipo en los objetivos, evitar retrasos y generar resultados de manera predecible (Schwaber, 2017).

### **ITERACIÓN E INCREMENTO**

El desarrollo en Scrum es incremental, lo que significa que el producto se construye en pequeños bloques funcionales que se entregan de forma frecuente. Cada iteración o Sprint produce un incremento del producto que puede ser utilizado y evaluado por los interesados (Beck et al., 2001).

### **ENFOQUE EN LA ENTREGA CONTINUA DE VALOR**

Más allá de cumplir con procesos, Scrum se centra en entregar un producto que aporte valor real al cliente o usuario final, asegurando que cada incremento pueda ser útil y represente un avance significativo hacia la visión del producto (Schwaber & Sutherland, 2020).

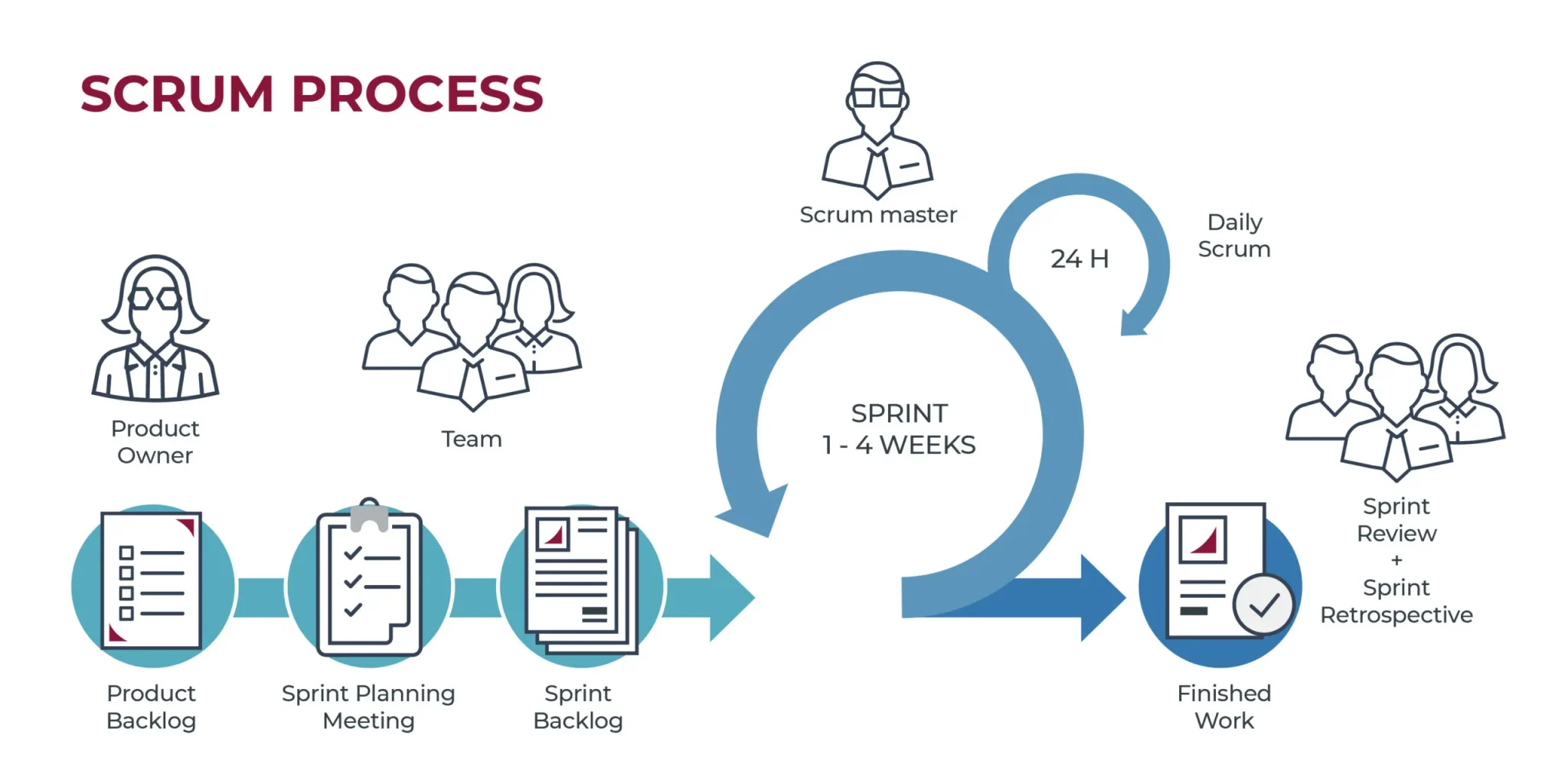


Figura 7. Dibujo del proceso del scrum(URL: <https://ausum.cloud/scrum-metodologia-agil-mas-popular-en-empresas/> )

* **COMPARATIVA:**

| **Metodología** | **Ventajas** | **Desventajas** |
| --- | --- | --- |
| Scrum | Iterativo, flexible, comunicación constante, alta adaptabilidad | Requiere disciplina y compromiso del equipo |
| Waterfall | Planificación detallada y medición de avances | Poco flexible, cambios costosos |
| Kanban | Visualización clara, flujo continuo de trabajo | Menor control sobre fechas y entregables |

Tabla 4. Comparación de metodologias (Elaboración propia)

**Beneficios en el proyecto:**

* Sprints de 2 semanas con entregables concretos.
* Product Owner prioriza tareas según valor de negocio.
* Incorporación continua de retroalimentación de usuarios.
* Garantiza desarrollo adaptado a necesidades reales y funcionales.

## **ROLES EN SCRUM**

Scrum define **tres roles principales** dentro del equipo:

1. **Product Owner (Propietario del Producto)**
   * Representa al cliente o los interesados (stakeholders).
   * Define la visión del producto.
   * Prioriza y gestiona el **Product Backlog** (lista de requisitos y tareas).
   * Su objetivo es **maximizar el valor** del producto.
2. **Scrum Master**
   * Es el facilitador del equipo Scrum.
   * Asegura que se comprendan y apliquen correctamente los principios de Scrum.
   * Elimina impedimentos que afecten al equipo.
   * Promueve la mejora continua.
3. **Development Team (Equipo de Desarrollo)**
   * Grupo multidisciplinario de entre 3 y 9 personas.
   * Se autoorganizan para entregar incrementos de producto en cada Sprint.
   * No tiene jerarquías internas; todos son responsables de la entrega.

## **ARTEFACTOS EN SCRUM**

Los artefactos son elementos que permiten **transparencia** y seguimiento del trabajo:

1. **Product Backlog**
   * Lista priorizada de requisitos, funcionalidades, mejoras y correcciones pendientes.
   * Evoluciona constantemente según las necesidades del producto.
2. **Sprint Backlog**
   * Subconjunto del Product Backlog seleccionado para un Sprint específico.
   * Incluye las tareas necesarias para completar el incremento del producto.
3. **Incremento del Producto**
   * Resultado final de un Sprint (software funcional).
   * Debe ser **usable y de calidad**, aunque no esté aún en producción.

## **REUNIONES EN SCRUM (EVENTOS)**

Scrum establece **eventos regulares** que marcan el ritmo del trabajo:

1. **Sprint**
   * Ciclo de trabajo de tiempo fijo (1 a 4 semanas).
   * Durante el Sprint se desarrolla un incremento de producto.
2. **Sprint Planning (Planificación del Sprint)**
   * Se define **qué se hará** en el Sprint y **cómo se hará.**
   * Participan todos los roles (Product Owner, Scrum Master y equipo).
3. **Daily Scrum (Reunión Diaria)**
   * Máximo 15 minutos cada día.
   * Cada miembro responde:
     + ¿Qué hice ayer?
     + ¿Qué haré hoy?
     + ¿Qué impedimentos tengo?
4. **Sprint Review (Revisión del Sprint)**
   * Al final del Sprint se presenta el incremento del producto a los stakeholders.
   * Se recopilan comentarios y feedback para el próximo ciclo.
5. **Sprint Retrospective (Retrospectiva del Sprint)**
   * Reunión interna del equipo tras la revisión.
   * Objetivo: identificar qué funcionó bien, qué se puede mejorar y definir acciones para el siguiente Sprint.

**2.4 ARQUITECTURA BACKEND CONCEPTUAL**

* **ARQUITECTURA BASADA EN MICROSERVICIOS:**
* Servicio de Inventario: Gestión de productos, stock y precios.
* Servicio de Ventas: Registro de transacciones y generación de facturas.
* Servicio de Usuarios: Autenticación, autorización y roles.
* Base de Datos PostgreSQL: Almacena información estructurada de manera segura.
* API Gateway: Punto de entrada unificado para solicitudes externas, distribuyendo las peticiones a cada microservicio.
* **BENEFICIOS:**
* Escalabilidad independiente por módulo.
* Mayor tolerancia a fallos.
* Despliegues modulares y rápidos.

**2.4.1 DIAGRAMAS CONCEPTUALES Y UML**

## DIAGRAMAS CONCEPTUALES

Un **diagrama conceptual** es una representación gráfica que organiza y relaciona **conceptos clave** de un tema.

* Sirve para **visualizar ideas** y mostrar cómo se conectan.
* Se utilizan nodos (conceptos) y enlaces (relaciones).
* Ayudan en la fase inicial del proyecto a **definir los elementos principales** (usuarios, productos, procesos, etc.) y cómo se relacionan.

En tu sistema de inventario, un diagrama conceptual podría mostrar:

* **Usuarios** (administrador, empleado, cliente).
* **Productos** (categorías, stock, precios).
* **Procesos** (entrada de productos, salida, actualización de stock, reportes).
* **Relaciones** entre ellos.

## UML (UNIFIED MODELING LANGUAGE)

**UML (Lenguaje Unificado de Modelado)** es un estándar de notación visual que se utiliza para **analizar, diseñar y documentar sistemas de software.**

* No es un lenguaje de programación, sino un conjunto de diagramas para representar la **arquitectura, procesos y comportamiento** de un sistema.
* Permite comunicar claramente cómo funcionará el sistema a nivel **estructural** y **dinámico.**

## DIAGRAMAS UML

## DIAGRAMAS ESTRUCTURALES

1. **Diagrama de Clases**
   * Muestra las entidades principales (Producto, Usuario, Inventario, Proveedor, Venta, Compra).
   * Define atributos (ej. stock, precio) y métodos (ej. actualizarStock()).
2. **Diagrama de Componentes (opcional, si tu proyecto es grande)**
   * Representa los módulos del sistema (ej. módulo de compras, ventas, reportes).

### DIAGRAMAS DE COMPORTAMIENTO

1. **Diagrama de Casos de Uso**
   * Representa los actores (Administrador, Empleado, Cliente) y sus interacciones con el sistema.
   * Ejemplo: “Registrar producto”, “Actualizar stock”, “Generar reporte”.
2. **Diagrama de Secuencia**
   * Muestra cómo interactúan los objetos en un proceso específico (ej. proceso de venta: usuario selecciona producto → sistema valida stock → actualiza inventario → genera comprobante).
3. **Diagrama de Actividades**
   * Representa el flujo de procesos dentro del sistema (ej. flujo de ingreso de mercadería: registrar producto → validar datos → actualizar stock → generar reporte).

* **DIAGRAMA DE FLUJO DE BACKEND:**

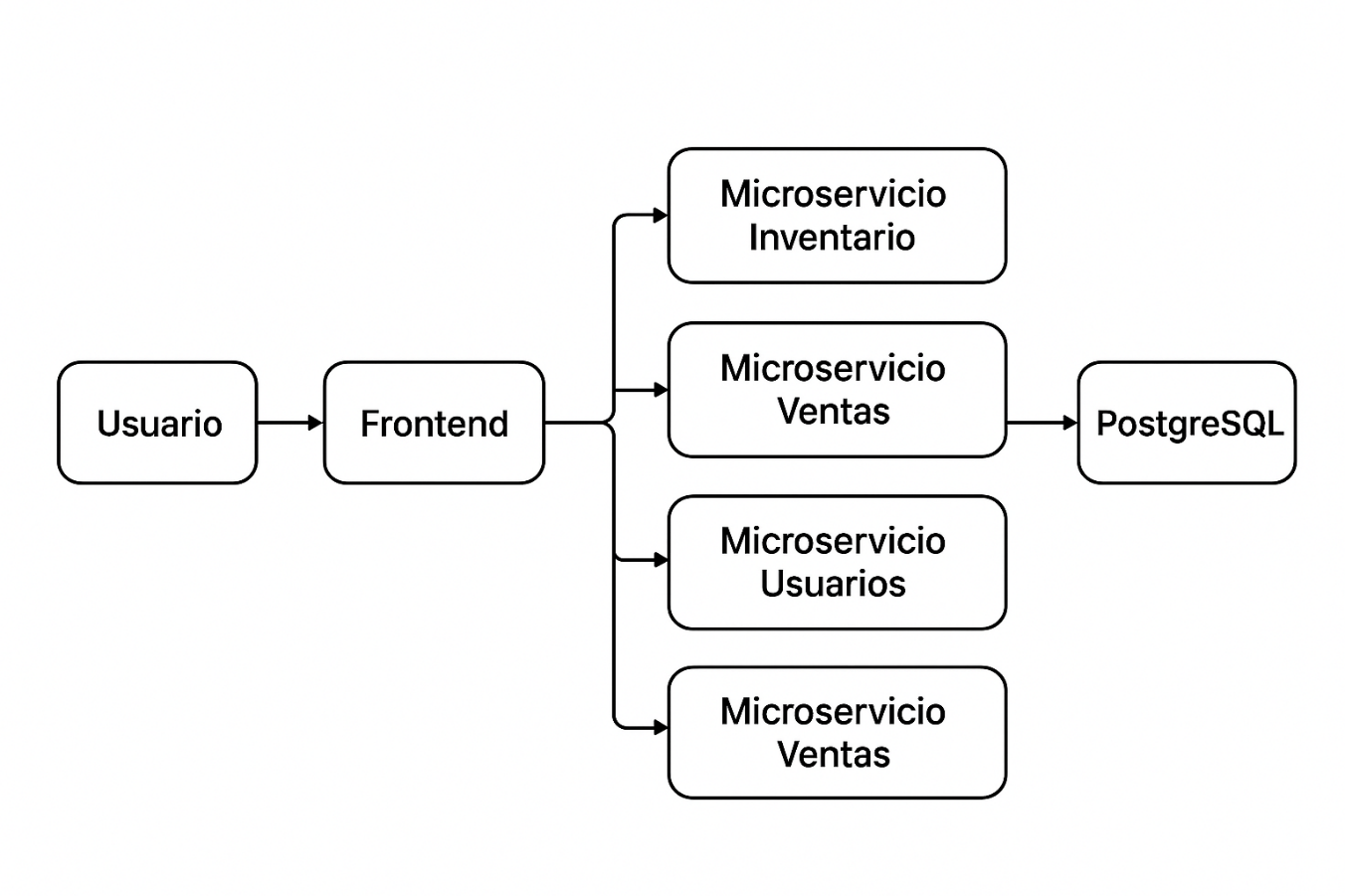


Figura 8. Ejemplo de diagrama de flujo(Elaboración propia)

* **DIAGRAMA DE CLASES UML SIMPLIFICADO:**

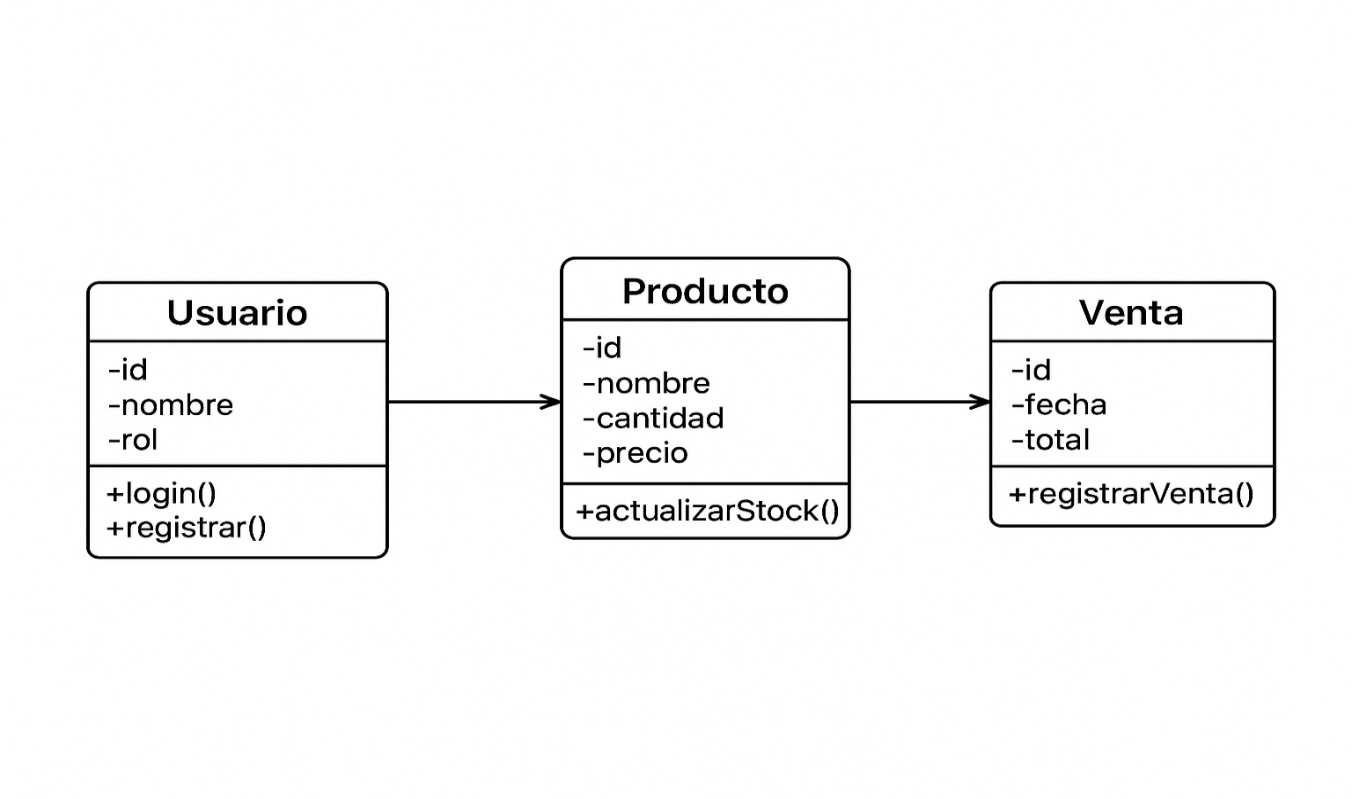


Figura 9. Ejemplo de diagrama de clase(Elaboración propia)

**CAPÍTULO 3**

**MARCO APLICATIVO**

**3.1 Introducción**

En este capítulo se presenta la propuesta de solución orientada al desarrollo de un sistema de gestión de ventas, cuyo objetivo principal es automatizar los procesos administrativos y operativos de la organización. Este sistema permitirá controlar de forma eficiente la información relacionada con personas, usuarios, distribuidores, productos, movimientos, kardex y documentos, mejorando la trazabilidad y la toma de decisiones dentro de la empresa.

La metodología seleccionada para el desarrollo es Scrum, debido a su enfoque ágil e iterativo, que favorece la adaptación a cambios en los requerimientos y garantiza la entrega de funcionalidades en ciclos cortos.  
El lenguaje de programación elegido es Python, por su versatilidad, legibilidad y ecosistema orientado al desarrollo backend. Como gestor de base de datos se utiliza PostgreSQL, un sistema robusto, estable y escalable que permite manejar grandes volúmenes de información de manera segura y eficiente.

La solución propuesta busca no solo optimizar el aspecto técnico, sino también el organizacional, estableciendo roles, responsabilidades y una planificación estructurada de los sprints. Esto asegura que el producto final cumpla con las expectativas de los usuarios y aporte valor real al proceso de gestión de ventas.

**3.2 Requerimientos Funcionales y No Funcionales**

Un requerimiento es una condición o capacidad que un sistema debe cumplir para ser aceptado por el usuario o cliente. Estos se clasifican en funcionales (qué hace el sistema) y no funcionales (cómo debe comportarse).

**3.2.1 Requerimientos Funcionales**

Los requerimientos funcionales definen las acciones y servicios que el sistema debe ofrecer al usuario.

Características principales:

* Están directamente relacionados con los objetivos del sistema.
* Son verificables mediante pruebas funcionales.
* Su ausencia afecta la funcionalidad esperada del sistema.

Requerimientos funcionales del sistema de gestión de ventas:

1. Gestión de personas: Registrar, modificar, eliminar y consultar la información básica de las personas vinculadas al sistema (clientes, empleados, distribuidores).
2. Gestión de usuarios: Crear, editar y eliminar usuarios del sistema, asignando roles y credenciales de acceso.
3. Gestión de roles: Administrar los diferentes niveles de acceso (administrador, cajero, distribuidor, etc.) y sus permisos correspondientes.
4. Gestión de distribuidores: Registrar y consultar información de los distribuidores, controlando su relación con los productos suministrados.
5. Gestión de productos: Controlar la información de productos, incluyendo nombre, descripción, precio, stock y categoría.
6. Gestión del kardex: Registrar las entradas y salidas de productos para mantener actualizado el inventario y garantizar trazabilidad.
7. Gestión de movimientos: Registrar y consultar los movimientos de ingreso y salida de productos, asociados al kardex.
8. Gestión de documentos: Generar y almacenar documentos relacionados con ventas, movimientos y distribución (por ejemplo, recibos o comprobantes).
9. Reportes: Emitir reportes generales y específicos sobre productos, movimientos, distribuidores y usuarios del sistema.

**3.2.2 Requerimientos No Funcionales**

Los requerimientos no funcionales establecen las condiciones de calidad y desempeño que debe cumplir el sistema.

Características principales:

* Evalúan atributos como rendimiento, seguridad o mantenibilidad.
* No describen acciones directas, sino criterios de calidad.
* Tienen gran impacto en la experiencia del usuario.

Requerimientos no funcionales del sistema:

1. Rendimiento: El sistema deberá responder a las consultas en menos de 2 segundos bajo carga estándar.
2. Escalabilidad: Permitirá la integración futura de módulos adicionales, como facturación electrónica o gestión avanzada de inventario.
3. Disponibilidad: El sistema deberá mantener una disponibilidad del 95% o superior.
4. Usabilidad: En caso de agregar una interfaz gráfica, esta deberá ser intuitiva, con menús accesibles y navegación simple.
5. Compatibilidad: El sistema podrá ejecutarse tanto en entornos Windows como Linux, gracias a la portabilidad de Python y PostgreSQL.
6. Seguridad: Las contraseñas se almacenarán cifradas y se implementarán controles de acceso por roles.
7. Mantenibilidad: Se aplicará programación modular y principios SOLID, facilitando la modificación o ampliación del sistema.

**3.3 Aplicación de la Metodología Scrum**

Scrum fue elegido por su enfoque ágil e incremental, permitiendo la entrega de versiones funcionales del sistema en cada sprint (Schwaber & Sutherland, 2020).

**3.3.1 Historias de Usuario**

* Administrador:  
  Como administrador, quiero registrar y administrar usuarios y roles, para controlar los permisos del sistema.
* Cajero:  
  Como cajero, quiero registrar movimientos y actualizaciones en el kardex, para mantener el inventario actualizado.
* Distribuidor:  
  Como distribuidor, quiero visualizar mis productos y documentos asociados, para gestionar mis entregas y pedidos.

**3.3.2 Product Backlog**

1. CRUD de personas, usuarios y roles.
2. CRUD de distribuidores.
3. CRUD de productos y control de stock.
4. Registro de movimientos y kardex.
5. Módulo de documentos (recibos, comprobantes, reportes).
6. Autenticación y seguridad de usuarios.
7. Generación de reportes administrativos.

**3.3.3 Sprint Backlog**

### **Sprint 1**

**• Configuración del entorno (Python + PostgreSQL):**Se realizó la preparación del entorno de desarrollo, instalando e integrando las herramientas principales. Python fue configurado como lenguaje base para el desarrollo del backend, mientras que PostgreSQL se estableció como sistema gestor de base de datos. Además, se configuraron las librerías necesarias para la conexión entre ambos entornos, garantizando la comunicación eficiente y segura entre la aplicación y la base de datos.

• **Creación de la base de datos y tablas:**  
En esta etapa se diseñó y creó la estructura inicial de la base de datos, definiendo las tablas, claves primarias, relaciones y restricciones. Se buscó garantizar la integridad referencial de los datos y una organización lógica que facilite las futuras ampliaciones del sistema.

• **Implementación del CRUD de personas y usuarios:**  
Se desarrollaron las operaciones básicas de creación, lectura, actualización y eliminación (CRUD) para las entidades persona y usuario. Esto permitió gestionar la información personal y las credenciales de acceso, estableciendo el módulo de autenticación y registro de usuarios dentro del sistema.

### **Sprint 2**

• **Implementación del módulo de productos:**  
Durante este punto se creó el módulo encargado de gestionar los productos disponibles en el sistema. Incluyó campos como nombre, código, descripción, unidad de medida y relación con el distribuidor. El objetivo fue permitir la administración eficiente del catálogo de productos y su actualización constante.

• **Gestión del kardex y movimientos:**  
Se implementó la lógica del módulo **kardex**, que registra los movimientos de entrada y salida de productos. Además, se desarrolló la tabla de **movimientos** para vincular cada transacción con un documento y usuario responsable. Este componente es clave para el control de inventarios, ya que permite monitorear las existencias en tiempo real.

**• Pruebas iniciales de inserción y consultas:**  
Se realizaron pruebas unitarias para verificar la correcta inserción de registros, así como consultas de verificación en la base de datos. Estas pruebas garantizaron la integridad de los datos y el correcto funcionamiento del módulo antes de avanzar a fases más complejas del desarrollo.

### **Sprint 3**

• **Gestión de documentos:**En este punto se implementó el módulo de documentos, permitiendo registrar los comprobantes que respaldan cada movimiento o transacción. Cada documento incluye su número correlativo y fecha, sirviendo como evidencia formal dentro del sistema.

• **Implementación de roles y permisos:**  
Se desarrolló la funcionalidad de control de accesos mediante la tabla **rol**, que define diferentes tipos de usuarios. Esto permitió asignar privilegios específicos, garantizando seguridad y delimitando las acciones disponibles según el perfil (administrador, cajero, distribuidor, etc.).

• **Generación de reportes básicos:**  
Se crearon reportes iniciales que permiten visualizar información general del sistema, tales como listados de productos, movimientos de inventario y registros de usuarios. Estos reportes facilitan el análisis y la toma de decisiones dentro de la gestión administrativa.

### **Sprint 4**

**• Pruebas finales del sistema:**  
En esta fase se realizaron pruebas integrales para verificar la funcionalidad completa del sistema. Se evaluó la interacción entre módulos, la consistencia de los datos y el cumplimiento de los requerimientos establecidos en etapas previas.

**• Optimización del rendimiento y la interfaz:**  
Se llevaron a cabo ajustes técnicos para mejorar la velocidad de respuesta, la eficiencia en las consultas a la base de datos y la experiencia del usuario. Además, se realizaron mejoras visuales en la interfaz para lograr una navegación más intuitiva y accesible.

**• Instalación y despliegue del sistema:**  
Finalmente, se procedió con la instalación del sistema en el entorno de producción. Se configuraron los parámetros necesarios para su ejecución estable y segura, asegurando su disponibilidad para el uso real por parte de los usuarios finales.

**3.3.4 Roles de Usuario**

* Product Owner: Representa los intereses de la organización, define prioridades y valida entregables. *En este proyecto, el Product Owner es Roberto Carlos García Mamani como propietario del negocio.*
* Scrum Master: Facilita el cumplimiento del marco de trabajo Scrum y elimina impedimentos. *En este proyecto, el Scrum Master es el ingeniero* Albino Martin *responsable como mi tutor encargado.*
* Equipo de desarrollo: Es mi persona el e0ncargado de la implementación técnica en Python y PostgreSQL, el diseño de la base de datos y la ejecución de pruebas.

**3.3.5 Planificación de Sprints**

#### **Sprint 1: Configuración de entorno y CRUD de usuarios**

En este primer sprint se llevó a cabo la configuración del entorno de desarrollo, incluyendo la instalación de herramientas, librerías y dependencias necesarias para el funcionamiento del proyecto. Asimismo, se desarrollaron las operaciones básicas de creación, lectura, actualización y eliminación (CRUD) del módulo de usuarios, permitiendo la gestión de credenciales y roles dentro del sistema. Este sprint sentó las bases para la estructura general del sistema y la administración de accesos.

#### **Sprint 2: Módulo de productos y kardex**

Durante el segundo sprint se implementó el módulo de **productos**, encargado del registro, actualización y control de los artículos manejados por la organización. Además, se desarrolló el módulo de **kardex**, el cual permite llevar un control detallado del inventario mediante el registro de movimientos de entrada y salida. Este sprint fue fundamental para garantizar la trazabilidad y gestión eficiente de los productos dentro del sistema.

#### **Sprint 3: Módulo de documentos y reportes**

En este sprint se diseñó e implementó el módulo de **documentos**, destinado al registro y control de comprobantes relacionados con las transacciones del sistema. También se desarrollaron las funcionalidades de **reportes**, orientadas a la generación de información consolidada sobre movimientos, inventarios y operaciones realizadas. Este avance permitió fortalecer la parte administrativa y analítica del sistema.

#### **Sprint 4: Pruebas finales, optimización e instalación del sistema**

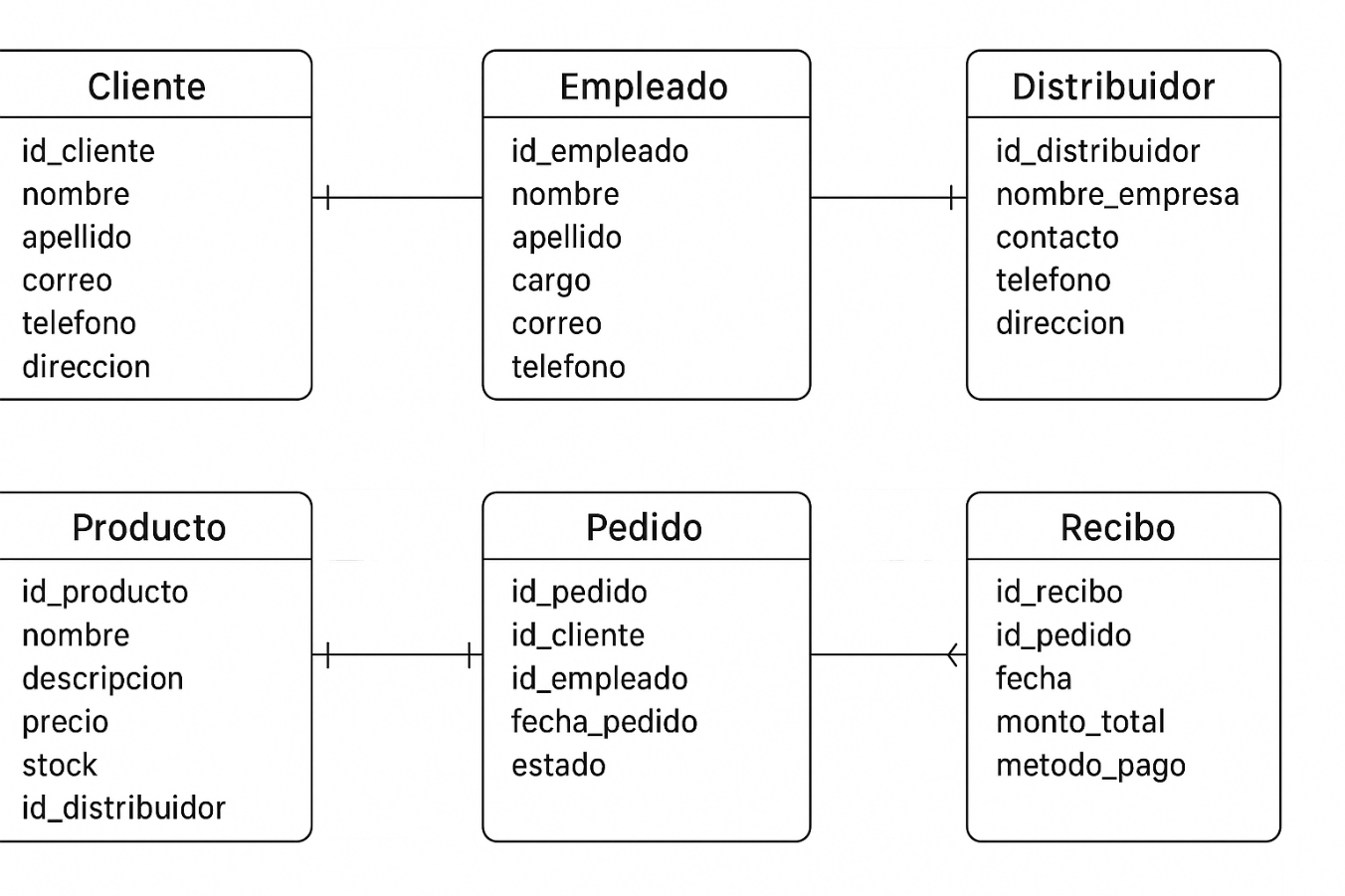
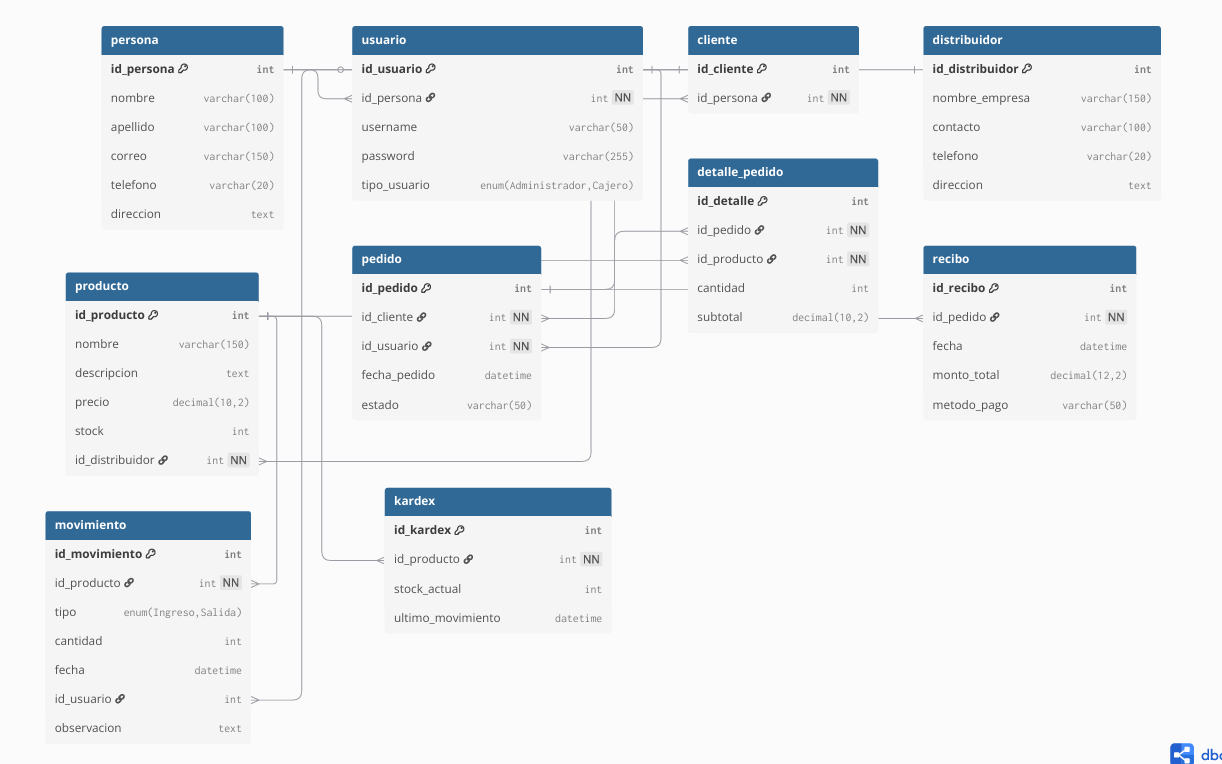
En el último sprint se realizaron las **pruebas finales** del sistema, verificando el correcto funcionamiento de cada módulo y la integración entre ellos. Se aplicaron procesos de **optimización** para mejorar el rendimiento y la usabilidad del sistema, corrigiendo posibles errores detectados durante las pruebas. Finalmente, se llevó a cabo la **instalación y despliegue** del sistema en el entorno de producción, completando así el ciclo de desarrollo del proyecto.

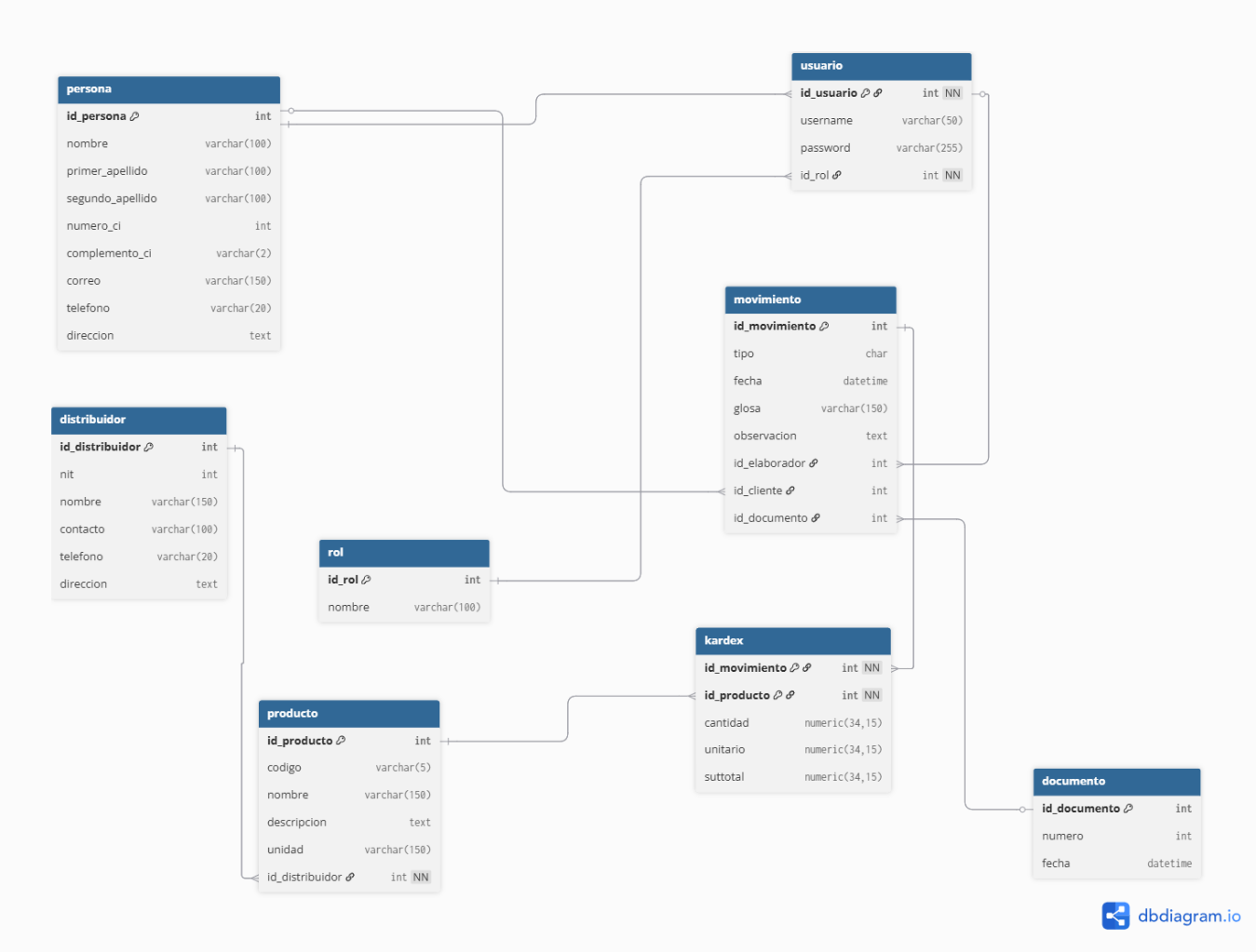
**3.3.6 Diseño de la Base de Datos**

**3.3.6.1 PostgreSQL**

El diseño de la base de datos se compone de ocho tablas principales:  
persona, usuario, rol, distribuidor, producto, kardex, movimiento y documento, todas relacionadas entre sí mediante claves foráneas.  
El modelo se encuentra normalizado y garantiza la integridad referencial. Además, PostgreSQL permite el uso de funciones almacenadas y triggers que automatizan procesos como la actualización de stock en el kardex o el registro de movimientos.

Versiones del sistema de base de datos:

* Versión 1: En la primera versión del sistema de base de datos se propuso una estructura inicial que contempla la distribución de las tablas junto con el contenido y los elementos principales que cada una de ellas debe incluir.
* Versión 2: En esta versión se realizaron modificaciones de gran alcance, reestructurando la base de datos y añadiendo nuevas tablas con el propósito de mejorar su comprensión, organización y administración.
* Versión 3: En esta versión se llevó a cabo un proceso de refinamiento y reestructuración de la base de datos, unificando y ajustando algunas tablas con el fin de optimizar su funcionamiento y facilitar la comprensión por parte del programador. Asimismo, se buscó garantizar que el diseño respondiera de manera adecuada a los requerimientos y necesidades establecidas.



**3.3.6.2 Diccionario de Datos**

El diccionario de datos describe los campos, tipos de datos y relaciones de cada tabla. Por ejemplo:

### **Tabla: PERSONA**

La tabla **persona** almacena la información básica de las personas que interactúan con el sistema, ya sea como usuarios, clientes o empleados. Contiene datos personales como nombres, apellidos, número de cédula de identidad, información de contacto y dirección. Esta tabla sirve como base para identificar a los diferentes actores del sistema y se relaciona con otras entidades, como usuario y movimiento, permitiendo un control más detallado de las acciones realizadas por cada individuo.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_PERSONA** | int | Identificador único de cada persona. Clave primaria. |
| **NOMBRE** | varchar(100) | Nombre de la persona. |
| **PRIMER\_APELLIDO** | varchar(100) | Primer apellido de la persona. |
| **SEGUNDO\_APELLIDO** | varchar(100) | Segundo apellido de la persona. |
| **NUMERO\_CI** | int | Número del documento de identidad de la persona. |
| **COMPLEMENTO\_CI** | varchar(2) | Complemento del número de cédula (si aplica). |
| **CORREO** | varchar(150) | Dirección de correo electrónico de la persona. |
| **TELEFONO** | varchar(20) | Número telefónico de contacto. |
| **DIRECCION** | text | Dirección domiciliaria de la persona. |

### **Tabla: USUARIO**

La tabla **usuario** tiene como función administrar el acceso al sistema mediante credenciales únicas. Cada registro corresponde a un usuario que puede iniciar sesión utilizando un nombre de usuario (username) y una contraseña (password) encriptada. Además, se vincula con la tabla **rol**, la cual define los permisos y el tipo de acceso asignado a cada usuario (por ejemplo, administrador, cajero o encargado). Esta tabla es esencial para el control de autenticación y seguridad del sistema.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_USUARIO** | int | Identificador único del usuario. Clave primaria. |
| **USERNAME** | varchar(50) | Nombre de usuario para el inicio de sesión. |
| **PASSWORD** | varchar(255) | Contraseña del usuario en formato encriptado. |
| **ID\_ROL** | int | Relación con la tabla rol, que define el tipo de usuario. Clave foránea. |

### **Tabla: ROL**

La tabla **rol** define los diferentes tipos de perfiles o niveles de acceso dentro del sistema. Cada rol representa un conjunto de permisos o responsabilidades asignadas a los usuarios. Por ejemplo, un administrador puede tener acceso total a la gestión de datos, mientras que un cajero podría estar limitado a operaciones de venta y registro de movimientos. Su relación directa con la tabla **usuario** permite una gestión eficiente de privilegios y jerarquías.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_ROL** | int | Identificador único del rol. Clave primaria. |
| **NOMBRE** | varchar(100) | Nombre del rol (por ejemplo, administrador, cajero, etc.). |

### **Tabla: DISTRIBUIDOR**

La tabla **distribuidor** registra la información de los proveedores o distribuidores que suministran los productos al sistema. Incluye datos como el número de identificación tributaria (NIT), nombre o razón social, contacto, teléfono y dirección. Su propósito principal es mantener un control organizado de las fuentes de abastecimiento, facilitando la trazabilidad de los productos y la gestión de relaciones comerciales.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_DISTRIBUIDOR** | int | Identificador único del distribuidor. Clave primaria. |
| **NIT** | int | Número de identificación tributaria del distribuidor. |
| **NOMBRE** | varchar(150) | Nombre o razón social del distribuidor. |
| **CONTACTO** | varchar(100) | Persona o medio de contacto principal. |
| **TELEFONO** | varchar(20) | Número telefónico del distribuidor. |
| **DIRECCION** | text | Dirección física del distribuidor. |

### **Tabla: DOCUMENTO**

La tabla **documento** tiene como finalidad registrar los comprobantes o respaldos documentales asociados a los movimientos de inventario, ventas o compras. Cada documento cuenta con un número correlativo y una fecha de emisión. Su relación con la tabla **movimiento** permite asociar los registros contables y de inventario con la documentación que los respalda, garantizando trazabilidad y control administrativo.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_DOCUMENTO** | int | Identificador único del documento. Clave primaria. |
| **NUMERO** | int | Número correlativo del documento. |
| **FECHA** | datetime | Fecha de emisión o registro del documento. |

### **Tabla: PRODUCTO**

La tabla **producto** almacena todos los artículos o bienes que se manejan dentro del sistema. Contiene información como código, nombre, descripción, unidad de medida y la relación con el distribuidor correspondiente. Esta tabla es fundamental para la gestión del inventario, ya que permite identificar cada producto, su origen y sus características principales. Además, se vincula con la tabla **kardex** para registrar las entradas y salidas de inventario.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_PRODUCTO** | int | Identificador único del producto. Clave primaria. |
| **CODIGO** | varchar(5) | Código interno de identificación del producto. |
| **NOMBRE** | varchar(150) | Nombre del producto. |
| **DESCRIPCION** | text | Descripción detallada del producto. |
| **UNIDAD** | varchar(150) | Unidad de medida del producto (por ejemplo, unidad, caja, litro, etc.). |
| **ID\_DISTRIBUIDOR** | int | Relación con la tabla distribuidor. Clave foránea. |

### **Tabla: MOVIMIENTO**

La tabla **movimiento** registra las operaciones de entrada y salida de productos en el sistema. Cada movimiento contiene información sobre su tipo (ingreso o salida), la fecha, una glosa descriptiva, observaciones, y las relaciones con las tablas **persona**, **documento** y **kardex**. Esta tabla cumple una función central dentro del sistema, ya que representa las transacciones que afectan el inventario y reflejan la actividad comercial o logística de la organización.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_MOVIMIENTO** | int | Identificador único del movimiento. Clave primaria. |
| **TIPO** | char | Tipo de movimiento (por ejemplo, ingreso o salida). |
| **FECHA** | datetime | Fecha en la que se realiza el movimiento. |
| **GLOSA** | varchar(150) | Descripción breve o referencia del movimiento. |
| **OBSERVACION** | text | Observaciones adicionales del movimiento. |
| **ID\_ELABORADOR** | int | Relación con la tabla persona que indica quién elaboró el movimiento. |
| **ID\_CLIENTE** | int | Relación con la tabla persona que identifica al cliente. |
| **ID\_DOCUMENTO** | int | Relación con la tabla documento. Clave foránea. |

### **Tabla: KARDEX**

La tabla **kardex** actúa como un registro histórico del inventario, detallando los movimientos de cada producto. Contiene campos que indican la cantidad, el precio unitario y el subtotal correspondiente a cada transacción. Está estrechamente vinculada con las tablas **producto** y **movimiento**, lo que permite llevar un control exacto de las existencias, las entradas y salidas de mercadería, y el valor económico del inventario. Su función es esencial para mantener actualizada la información de stock y generar reportes de control.

| **CAMPO** | **TIPO DE DATO** | **DESCRIPCIÓN** |
| --- | --- | --- |
| **ID\_MOVIMIENTO** | int | Relación con la tabla movimiento. Clave foránea. |
| **ID\_PRODUCTO** | int | Relación con la tabla producto. Clave foránea. |
| **CANTIDAD** | numeric(34,15) | Cantidad de productos registrados en el movimiento. |
| **UNITARIO** | numeric(34,15) | Precio unitario del producto. |
| **SUBTOTAL** | numeric(34,15) | Subtotal calculado según cantidad y precio unitario. |

**3.4 Fase de Construcción**

La **fase de construcción** es la etapa del ciclo de desarrollo de software en la que se **implementa y codifica la solución propuesta**, transformando los diseños y especificaciones previas en un sistema funcional.

En el contexto de un **sistema de inventario,** esta fase consiste en **desarrollar los módulos y componentes del software** que permiten gestionar los productos, movimientos de entrada y salida, proveedores, usuarios y reportes, entre otros.

**3.4.1 Herramientas de Desarrollo**

* Python 3.13.5: Lenguaje principal. Ver la versión del lenguaje que utilizo
* FastAPI: Framework para crear APIs RESTful rápidas y seguras.
* PostgreSQL 15: Gestor de base de datos……
* PgAdmin 4: Herramienta para administración gráfica de PostgreSQL.

**3.4.2 Herramientas de Diseño**

* Draw.io: Para diagramas entidad-relación.
* PlantUML: Para los diagramas UML.

**3.4.3 Herramientas de Prueba**

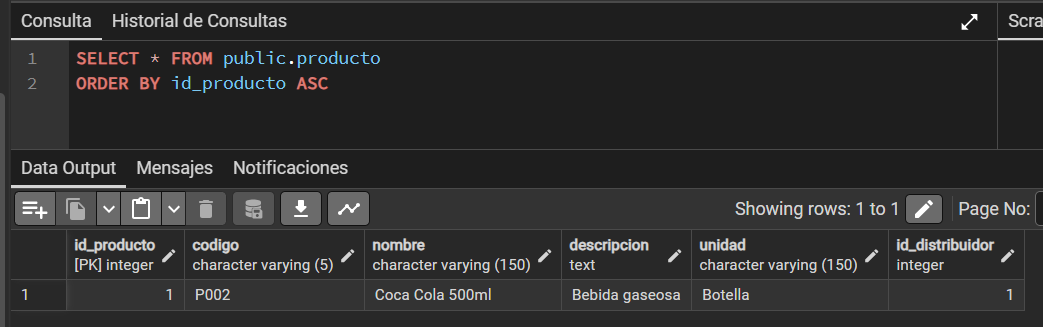
* PyTest: Para pruebas unitarias en Python.
* Postman: Para pruebas de endpoints REST.
* PgAdmin4: Para consultas SQL.

**3.5 Fase de Pruebas**

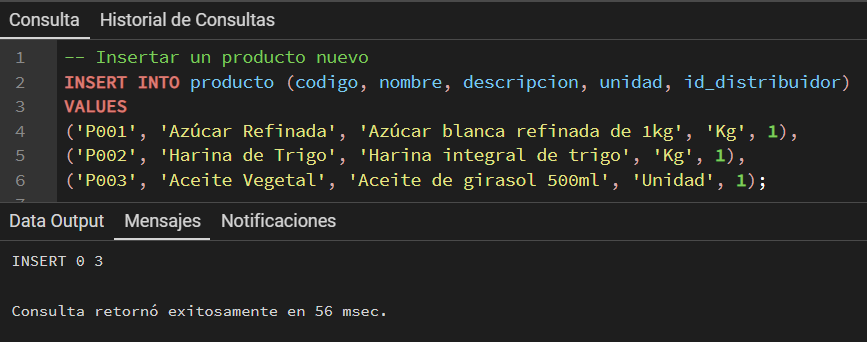
1. Unitarias: Validación de funciones CRUD.

Tabla producto:

En esta consulta SQL se puede mostrar los datos que existen en la tabla producto desde la consola de la base de datos de PostgreSQL, confirmando así que los datos se guardan correctamente.



En esta consulta SQL se puede mostrar la inserción de los nuevos datos que se añadirán en la tabla producto desde la consola de la base de datos PostgreSQL, confirmando así que los datos se ingresan correctamente.



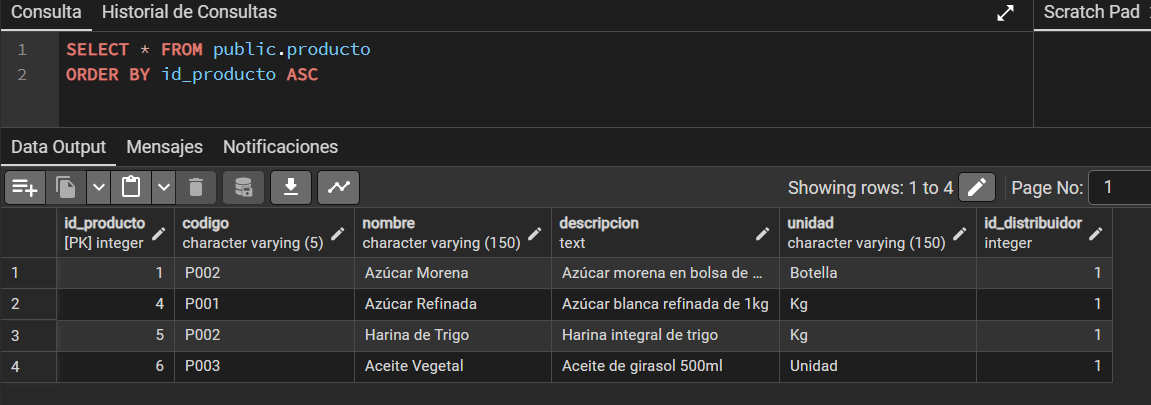
En esta consulta SQL se puede mostrar los datos que existen en la tabla producto desde la consola de la base de datos PostgreSQL, confirmando así que los datos que se ingresan se guardan correctamente.



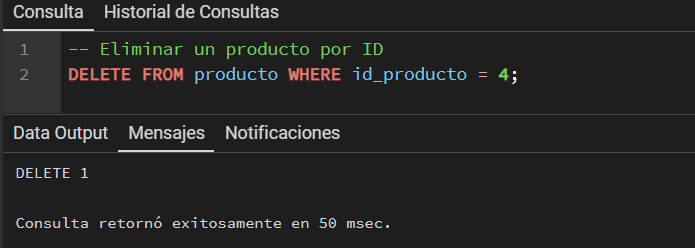
En esta consulta SQL se puede mostrar la actualización de datos que se añadirán en la tabla producto desde la consola de la base de datos PostgreSQL, confirmando así que los datos se actualizan correctamente.



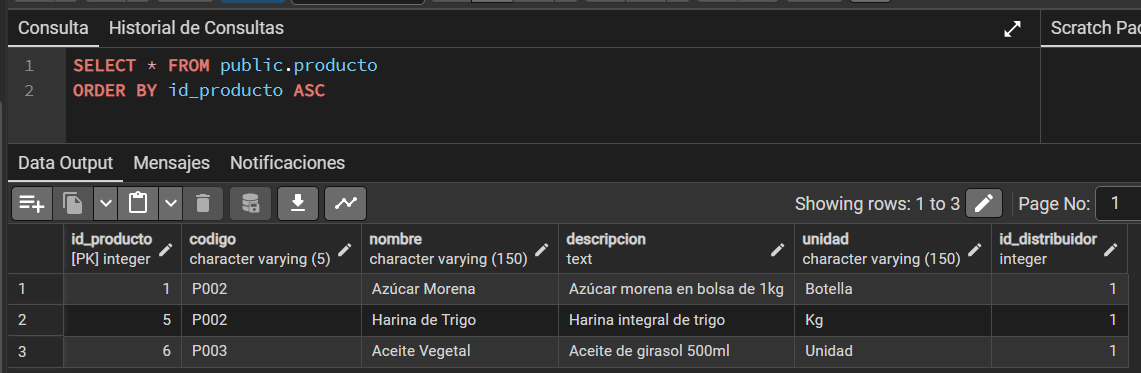
En esta consulta SQL se puede mostrar que la actualización de los datos que se añadieron en la tabla producto desde la consola de la base de datos PostgreSQL, confirmando así que los datos se actualizaron correctamente.



En esta consulta SQL se puede mostrar la eliminación de datos que existen en la tabla producto desde la consola de la base de datos PostgreSQL, confirmando así que los datos se eliminan correctamente.



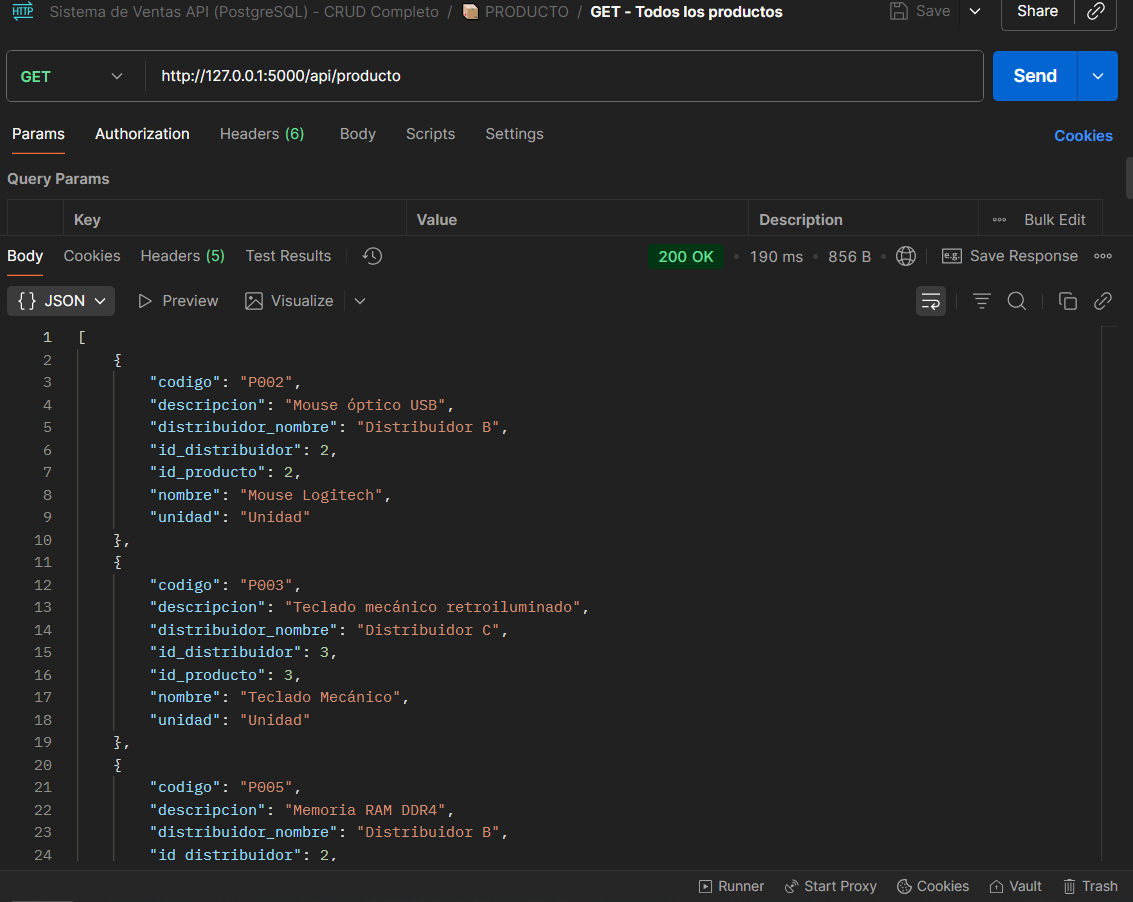
En esta consulta SQL se puede mostrar que la eliminación del dato en la tabla producto desde la consola de la base de datos PostgreSQL, confirmando así que los datos se eliminan correctamente.



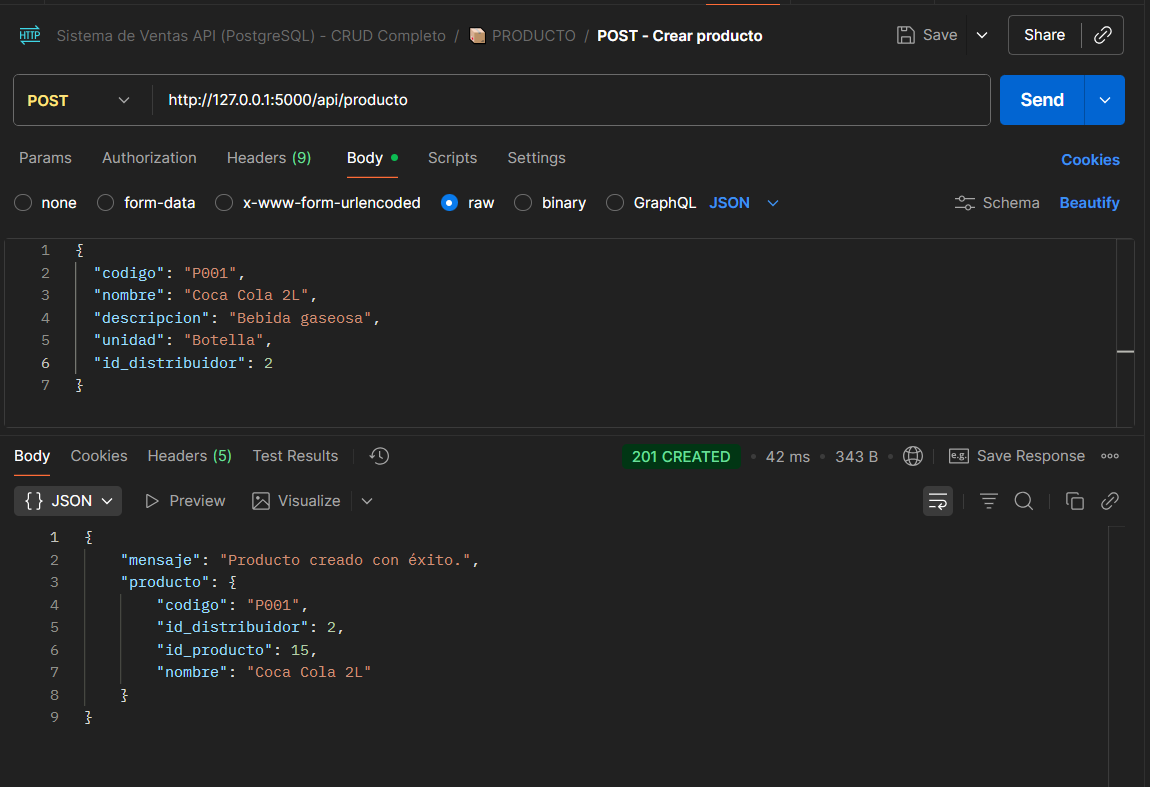
1. Integración: Verificación de la comunicación entre Python y PostgreSQL, teniendo como intermediario a Postman.

Producto:

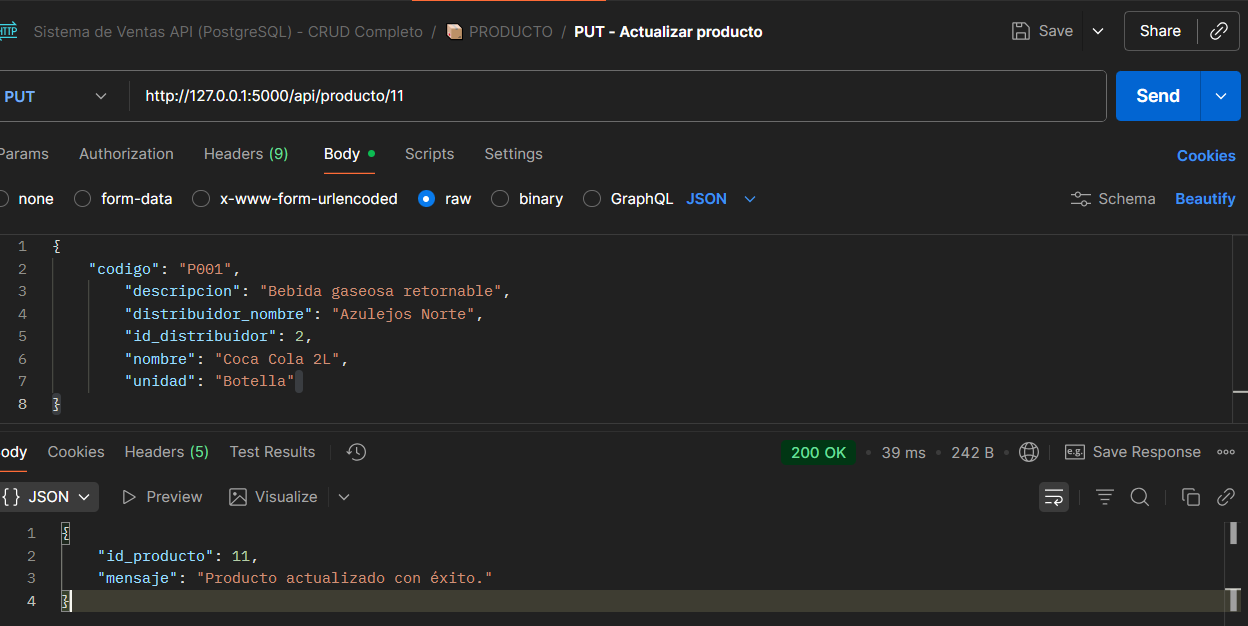
En esta sección se muestra como postman hace la petición de información de toda la tabla producto mediante el método GET, el cual le devolverá si tiene una buena conexión todos los datos que estén en la tabla producto.



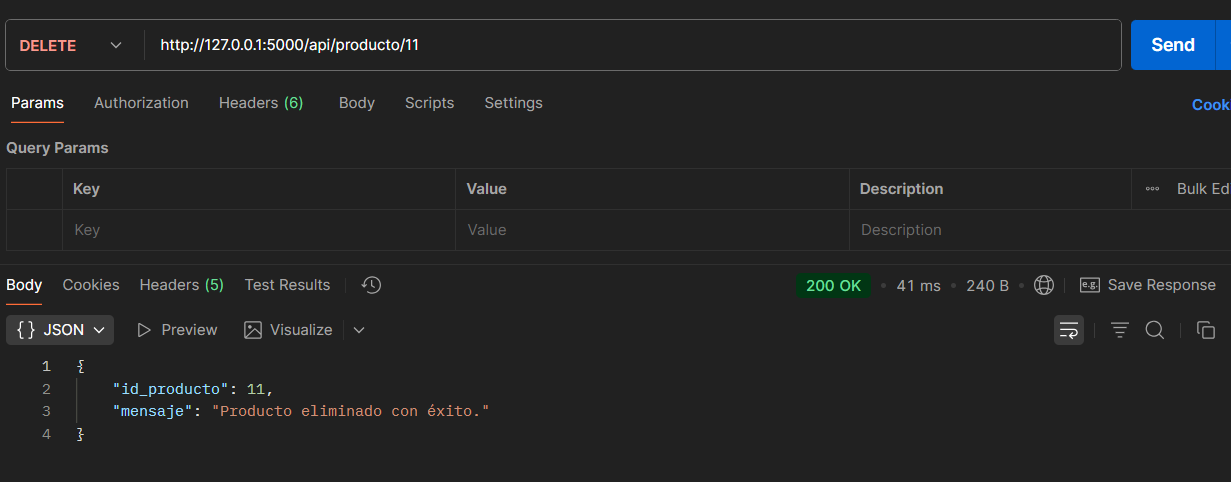
En esta sección se muestra como postman hace el envió de un JSON con un nuevo producto para la tabla producto mediante el método POST, el cual le devolverá si tiene una buena conexión la inserción correcta del producto.



En esta sección se muestra como postman hace el envió de un JSON con la actualización del producto para la tabla producto mediante el método PUT, el cual le devolverá si tiene una buena conexión la actualización correcta del producto.

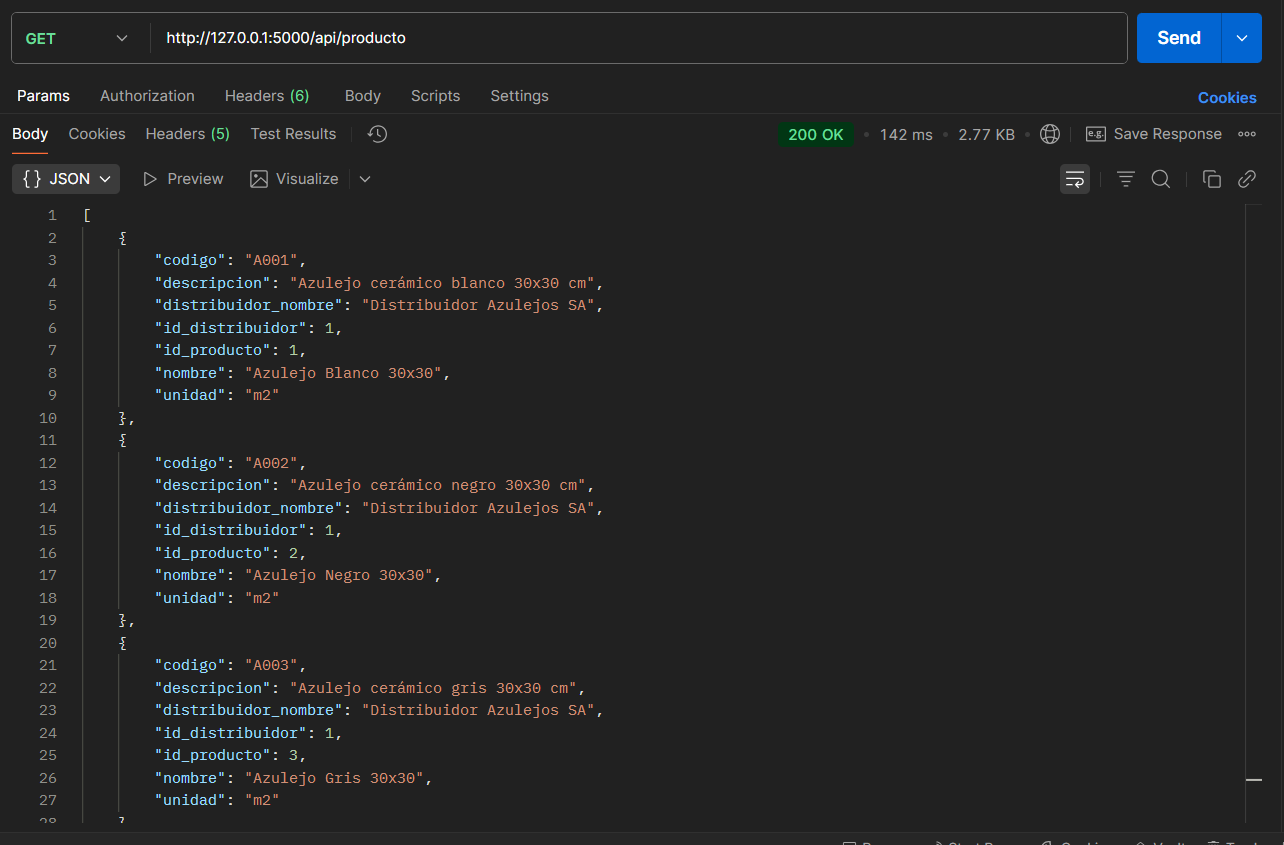


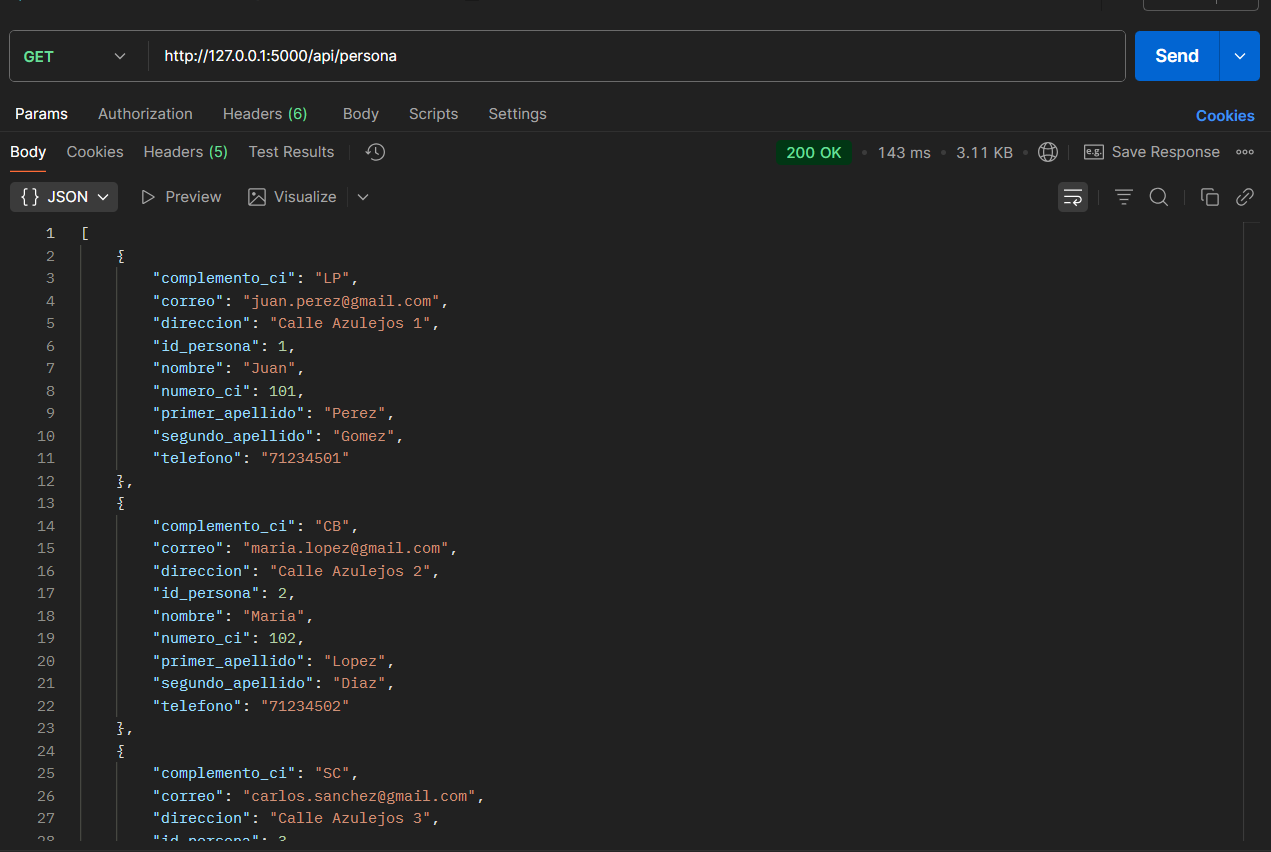
En esta sección se muestra como postman hace el envió de un JSON con la eliminación del producto para la tabla producto mediante el método DELETE, el cual le devolverá si tiene una buena conexión la eliminación correcta del producto.

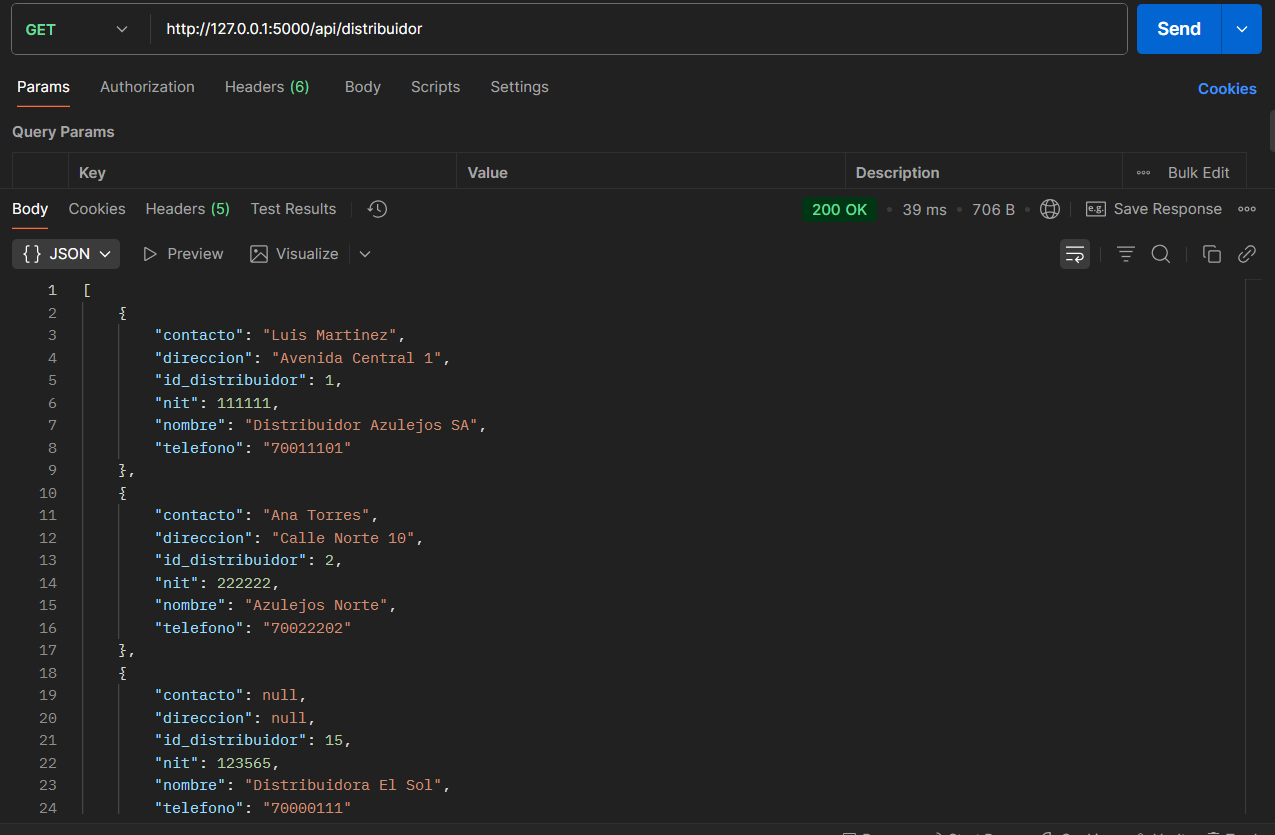


1. Aceptación: Validación final con los usuarios y revisión de los procesos del sistema.

En las siguientes imágenes se mostraran que los cruds funcionan correctamente.







**3.6 Instalación del Sistema**

El sistema será instalado en un servidor con Ubuntu Server 20.04 LTS.  
La base de datos PostgreSQL se configurará con roles y permisos adecuados.  
El backend en Python se desplegará con Uvicorn o Gunicorn, pudiendo emplear Docker para mejorar la portabilidad.

Pasos principales:

1. Creación de la base de datos y tablas.
2. Configuración de variables de entorno.
3. Despliegue de la API.
4. Configuración de seguridad (cifrado y firewall).

**CAPÍTULO 4**

**CONCLUSIONES Y RECOMENDACIONES**

**4.1 CONCLUSIONES**

**4.1.1 CONCLUSIONES ACERCA DE LA METODOLOGÍA**

La aplicación de la metodología Scrum demostró ser una estrategia adecuada para el desarrollo del sistema de gestión de ventas.  
Su enfoque ágil e incremental permitió estructurar el trabajo en sprints, entregando versiones funcionales del sistema en etapas definidas y adaptándose de forma flexible a los cambios surgidos durante el proceso.  
Gracias a la colaboración entre los diferentes roles —Product Owner, Scrum Master y equipo de desarrollo—, se logró una comunicación fluida, un mejor control del progreso y una gestión eficiente del tiempo.  
En conclusión, el uso de Scrum garantizó la obtención de un producto final más alineado con los requerimientos reales de la organización, cumpliendo con los plazos y la calidad esperada.

**4.1.2 CONCLUSIONES ACERCA DE LAS HERRAMIENTAS**

Las herramientas seleccionadas demostraron ser eficientes y adecuadas para los objetivos del proyecto:

* Python brindó un entorno de desarrollo flexible, con librerías que facilitaron la implementación del backend y la conexión con la base de datos.
* FastAPI permitió construir una API moderna, segura y de alto rendimiento, optimizando el manejo de peticiones HTTP.
* PostgreSQL ofreció un control robusto de datos, asegurando la integridad referencial entre las tablas persona, usuario, rol, distribuidor, producto, kardex, movimiento y documento.
* PgAdmin 4 simplificó la administración y pruebas sobre la base de datos, mientras que Postman permitió verificar los endpoints del sistema.

En conjunto, estas herramientas posibilitaron un desarrollo ordenado, escalable y de fácil mantenimiento.

**4.1.3 CONCLUSIONES ACERCA DE LOS OBJETIVOS**

Los objetivos planteados al inicio del proyecto fueron cumplidos satisfactoriamente.  
El sistema desarrollado logró automatizar los procesos de gestión de ventas e inventario, permitiendo un control eficiente de los productos, movimientos, kardex y documentos asociados.  
Además, se implementó una estructura de usuarios y roles que fortalece la seguridad y organización de la información.  
El cumplimiento de los objetivos específicos —como la creación del CRUD, el control de stock, la trazabilidad de movimientos y la generación de reportes— evidencia que la solución propuesta responde adecuadamente a las necesidades detectadas en la fase de análisis.

**4.2 RECOMENDACIONES**

**4.2.1 RECOMENDACIONES ACERCA DEL USO DEL SISTEMA**

Se recomienda que el personal encargado del uso del sistema reciba una capacitación básica sobre las funciones principales, como el registro de usuarios, control de productos y consulta de movimientos.  
Asimismo, se sugiere establecer roles definidos (administrador, cajero, distribuidor) para garantizar que cada usuario acceda únicamente a las funciones que le competen.  
Es importante realizar copias de seguridad periódicas de la base de datos para prevenir la pérdida de información ante posibles fallos del sistema.

**4.2.2 RECOMENDACIONES ACERCA DE LA INSTALACIÓN Y CONFIGURACIÓN DEL SISTEMA**

Para su correcta instalación, se recomienda que el sistema sea implementado en un servidor con Ubuntu Server 20.04 LTS o superior, con PostgreSQL y Python actualizados.  
Durante la configuración, deben establecerse las variables de entorno correspondientes (host, puerto, usuario, contraseña y nombre de la base de datos), así como aplicar las políticas de seguridad adecuadas:

* Cifrado de contraseñas.
* Configuración de cortafuegos.
* Limitación de accesos externos no autorizados.

Además, se sugiere documentar el proceso de instalación para futuras implementaciones o migraciones.

**4.2.3 RECOMENDACIONES ACERCA DE LAS ACTUALIZACIONES DEL SISTEMA**

Se recomienda mantener el sistema en constante actualización, tanto en su código como en las herramientas que lo soportan.  
Las futuras versiones podrían incluir módulos adicionales como:

* Facturación electrónica.
* Control avanzado de inventario.
* Reportes gráficos interactivos.
* Interfaz web o móvil para mejorar la experiencia del usuario final.

Asimismo, se aconseja realizar mantenimientos preventivos en la base de datos y actualizar las dependencias de Python y FastAPI para garantizar la seguridad, estabilidad y compatibilidad del sistema a largo plazo.

**Bibliografías**

**2.6 REFERENCIAS BIBLIOGRÁFICAS**

## INGENIERÍA DE SOFTWARE Y METODOLOGÍAS

* Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Manifesto for Agile Software Development. Agile Alliance. Recuperado de <https://agilemanifesto.org>
* Cohn, M. (2004). User Stories Applied: For Agile Software Development. Addison-Wesley.
* Pressman, R. S. (2010). Ingeniería del software: un enfoque práctico (7.ª ed.). McGraw-Hill.
* Pressman, R. S. (2014). Ingeniería del software: un enfoque práctico. McGraw-Hill.
* Schwaber, K. (2017). Agile Project Management with Scrum. Microsoft Press.
* Schwaber, K., & Sutherland, J. (1995). SCRUM Software Development Process. OOPSLA Business Object Design and Implementation Workshop.
* Schwaber, K., & Sutherland, J. (2020). The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org.
* Sommerville, I. (2011). Ingeniería del software. Addison-Wesley.
* Takeuchi, H., & Nonaka, I. (1986). The new new product development game. Harvard Business Review, 64(1), 137–146.

## PYTHON Y FRAMEWORKS

* Grinberg, M. (2018). Flask Web Development. O’Reilly Media.
* Lutz, M. (2013). Learning Python (5th ed.). O’Reilly Media.
* Python Software Foundation. (2001). Python Software Foundation: About. Recuperado de <https://www.python.org/psf/>
* Python Software Foundation. (2023). Python Documentation. Recuperado de <https://www.python.org/doc/>
* Ramírez, S. (2020). FastAPI: High Performance Python Framework. FastAPI Docs. Recuperado de <https://fastapi.tiangolo.com/>
* TIOBE Index. (2023). TIOBE Programming Community Index. Recuperado de <https://www.tiobe.com/tiobe-index/>
* Van Rossum, G. (1991). Python Tutorial. CWI Report CS-R9526.
* Van Rossum, G. (2010). History of Python. Python.org. Recuperado de <https://www.python.org/doc/essays/>
* Van Rossum, G., & Drake, F. L. (2009). The Python Language Reference Manual (Release 3.0). Python Software Foundation.
* FastAPI. (s.f.). Features. Recuperado de [https://fastapi.tiangolo.com/features/](https://fastapi.tiangolo.com/features/?utm_source=chatgpt.com)
* Turing. (2023). Flask vs FastAPI: A Detailed Comparison. Recuperado de [https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison](https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison?utm_source=chatgpt.com)
* GeeksforGeeks. (2023). Flask vs. FastAPI: Which One to Choose. Recuperado de [https://www.geeksforgeeks.org/blogs/flask-vs-fastapi/](https://www.geeksforgeeks.org/blogs/flask-vs-fastapi/?utm_source=chatgpt.com)
* BetterStack. (2023). Flask vs FastAPI: An In-Depth Framework Comparison. Recuperado de [https://betterstack.com/community/guides/scaling-python/flask-vs-fastapi/](https://betterstack.com/community/guides/scaling-python/flask-vs-fastapi/?utm_source=chatgpt.com)
* Imaginary Cloud. (2023). Flask vs FastAPI: what's better for Python app development?. Recuperado de [https://www.imaginarycloud.com/blog/flask-vs-fastapi](https://www.imaginarycloud.com/blog/flask-vs-fastapi?utm_source=chatgpt.com)

## POSTGRESQL Y BASES DE DATOS

* Momjian, B. (2015). PostgreSQL: Up and Running (3rd ed.). O’Reilly Media.
* PostgreSQL Global Development Group. (2023). PostgreSQL Documentation. Recuperado de <https://www.postgresql.org/docs/>
* Stonebraker, M., Rowe, L. (1986). The design of POSTGRES. Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, 340–355.
* Stonebraker, M., Rowe, L., Hirohama, M., & Wong, H. (1991). The implementation of POSTGRES. IEEE Transactions on Knowledge and Data Engineering, 2(1), 125–142.
* Stonebraker, M., et al. (1986). The POSTGRES data model. ACM SIGMOD.

## GIT, GITHUB Y SOFTWARE LIBRE

* Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, 1277–1286.
* Fogel, K. (2017). Producing Open Source Software: How to Run a Successful Free Software Project. O’Reilly Media.
* GitHub. (2023). The State of the Octoverse. GitHub, Inc. Recuperado de <https://octoverse.github.com>
* Loeliger, J., & McCullough, M. (2012). Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development (2nd ed.). O’Reilly Media.
* Microsoft. (2018). Microsoft to acquire GitHub for $7.5 billion. Microsoft News Center. Recuperado de <https://news.microsoft.com>
* Torvalds, L. (2005). Git – Fast Version Control System. Linux Kernel Archives.

## ARQUITECTURA DE SOFTWARE

* Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.
* UML 2.5 Diagrams Overview – UML Diagrams <https://www.uml-diagrams.org/uml-25-diagrams.html>
* Unified Modeling Language (UML) Introduction – GeeksforGeeks <https://www.geeksforgeeks.org/system-design/unified-modeling-language-uml-introduction/>
* Learn About All Types of UML Diagrams with Examples – Creately <https://creately.com/blog/diagrams/uml-diagram-types-examples/>
* overview - where and why each UML diagram is used – draw.io <https://www.drawio.com/blog/uml-overview>

**Bibliografías**

* “Functional vs. Non Functional Requirements”, GeeksforGeeks [GeeksforGeeks+1](https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/?utm_source=chatgpt.com)
* “Requerimientos funcionales vs no funcionales” (Visure Solutions) [Visure Solutions+2Visure Solutions+2](https://visuresolutions.com/es/gu%C3%ADa-de-trazabilidad-de-gesti%C3%B3n-de-requisitos/requisitos-funcionales-vs-no-funcionales/?utm_source=chatgpt.com)
* “Nonfunctional Requirements: Examples, Types and Approaches” (AltexSoft) [AltexSoft](https://www.altexsoft.com/blog/non-functional-requirements/?utm_source=chatgpt.com)
* “Nonfunctional requirement” (Wikipedia) [Wikipedia](https://en.wikipedia.org/wiki/Non-functional_requirement?utm_source=chatgpt.com)
* Otras fuentes como Mad Devs, Ironhack, etc. [Custom Software Development Company+2Ironhack+2](https://maddevs.io/blog/functional-vs-non-functional-requirements/?utm_source=chatgpt.com)
* Karl E. Wiegers, *Software Requirements* (Microsoft Press) — trata ampliamente la ingeniería de requisitos.
* Bernd Bruegge y Allen H. Dutoit, *Object-Oriented Software Engineering* — incluye discusión de requisitos y atributos de calidad.
* IEEE Standard 830 (o sus actualizaciones) — estándar para especificación de requerimientos de software.
* Scott Ambler, *Technical (Non-Functional) Requirements: An Agile Introduction* — para enfoques ágiles en requisitos no funcionales.
* Universidad Nacional del Sur. (2013). Ingeniería de Requerimientos I. Departamento de Ciencias e Ingeniería de la Computación.  
  Recuperado de: [https://cs.uns.edu.ar/~td/ayds2013/downloads/Teoria/ADyS\_2013\_04%20-%20Ingenieria\_de\_Requerimientos\_I.pdf](https://cs.uns.edu.ar/~td/ayds2013/downloads/Teoria/ADyS_2013_04%20-%20Ingenieria_de_Requerimientos_I.pdf?utm_source=chatgpt.com)
* GeeksforGeeks. (2024). Functional vs Non-Functional Requirements.  
  Recuperado de: [https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/](https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/?utm_source=chatgpt.com)
* Visure Solutions. (2024). Requerimientos no funcionales: definición, ejemplos y clasificación.  
  Recuperado de: [https://visuresolutions.com/es/gu%C3%ADa-de-limosna/requerimientos-no-funcionales/](https://visuresolutions.com/es/gu%C3%ADa-de-limosna/requerimientos-no-funcionales/?utm_source=chatgpt.com)
* Ironhack. (2023). Functional vs Non-Functional Requirements: Understanding the Core Differences.  
  Recuperado de: [https://www.ironhack.com/us/blog/functional-vs-non-functional-requirements-understanding-the-core-differences-and](https://www.ironhack.com/us/blog/functional-vs-non-functional-requirements-understanding-the-core-differences-and?utm_source=chatgpt.com)
* Wikipedia. (2024). Non-functional requirement.  
  Recuperado de: [https://en.wikipedia.org/wiki/Non-functional\_requirement](https://en.wikipedia.org/wiki/Non-functional_requirement?utm_source=chatgpt.com)

Clockwise Software. (2024). Functional and Non-Functional Requirements: Examples and Differences.  
Recuperado de: [https://clockwise.software/blog/functional-and-nonfunctional-requirements/](https://clockwise.software/blog/functional-and-nonfunctional-requirements/?utm_source=chatgpt.com)

**ANEXOS**

1. **TÉCNICAS DE RECOLECCIÓN DE INFORMACIÓN APLICADAS**
2. **IMÁGENES RELACIONADAS A SU PROYECTO**
3. **GLOSARIO DETÉRMINOS TÉCNICOS (SI CORRESPONDE)**
4. **MANUAL DE USUARIO**
5. **MANUAL TECNICO**