UNIVERSIDAD NACIONAL AGRARIA DE LA SELVA

FACULTAD DE INGENIERIA EN INFORMATICA Y SISTEMAS

ESCUELA PROFESIONAL DE INGENIERIA EN INFORMATICA Y SISTEMAS



***“Sistema de Gestión de Ventas en Línea (Arquitectura en Capas)”***

**Proyecto:** Documento Funcional

**Curso:** Arquitectura de Software

**Docente:** Ing. Christian García Villegas

**Integrantes:**

* Dairon Said Hidalgo Robles
* Sergio Andrés Henríquez Moya
* Junior Yonel Ordoñez Silva
* Brandon Gabriel Benavides Vargas

Tingo María - Perú

noviembre 2025

**INDICE**

[1 Introducción 1](#_Toc213275852)

[1.1 Descripción general del sistema 1](#_Toc213275853)

[1.2 Objetivos del sistema 2](#_Toc213275854)

[1.3 Motivación de la elección de la arquitectura 3](#_Toc213275855)

[2 Análisis de Requisitos 4](#_Toc213275856)

[2.1 Historias de Usuario 5](#_Toc213275857)

[2.2 Requisitos Funcionales 25](#_Toc213275858)

[2.2.1 Modularización de Requisitos Funcionales 26](#_Toc213275859)

[2.2.1.1 Gestión de Usuarios 26](#_Toc213275860)

[2.2.1.2 Gestión de Productos 26](#_Toc213275861)

[2.2.1.3 Ventas o Facturación 26](#_Toc213275862)

[2.2.1.4 Reportes y Analíticas 27](#_Toc213275863)

[2.2.1.5 Notificaciones 27](#_Toc213275864)

[2.2.1.6 Configuración 27](#_Toc213275865)

[2.2.1.7 Interfaces de Integración 27](#_Toc213275866)

[2.3 Requisitos No Funcionales 28](#_Toc213275867)

[2.3.1 Categorización de Requisitos No Funcionales Según la ISO 25010 29](#_Toc213275868)

[3 Selección y Descripción de la Arquitectura 30](#_Toc213275869)

[3.1 Descripción General de la Arquitectura en Capas 31](#_Toc213275870)

[3.1.1 Capa de Presentación (Front/Interface Layer) 31](#_Toc213275871)

[3.1.2 Capa de Lógica de Negocio (Business Logic Layer) 31](#_Toc213275872)

[3.1.3 Capa de Acceso a Datos (Data Access Layer) 32](#_Toc213275873)

[3.2 Flujo de Comunicación entre Capas 33](#_Toc213275874)

[3.3 Justificación de la Elección 34](#_Toc213275875)

[A. Razones Técnicas 34](#_Toc213275876)

[B. Razones de Negocio y Académicas 35](#_Toc213275877)

[3.4 Tecnologías y Herramientas Empleadas 36](#_Toc213275878)

[4 Diseño Preliminar 37](#_Toc213275879)

[4.1 Diagrama General del Sistema (UML de Capas) 37](#_Toc213275880)

[4.2 Diagrama de Componentes 38](#_Toc213275881)

[4.3 Diagrama de Clases 39](#_Toc213275882)

[5 Plan de Trabajo 40](#_Toc213275883)

[5.1 Cronograma de Actividades 40](#_Toc213275884)

[5.2 Próximos Pasos 42](#_Toc213275885)

[6 Conclusiones 43](#_Toc213275886)

# Introducción

## ****Descripción general del sistema****

El presente proyecto consiste en el desarrollo de un **Sistema de Gestión de Ventas para una tienda en línea**, cuyo propósito es facilitar la administración de productos, el registro de ventas y la generación de reportes. El sistema permitirá que los administradores gestionen el catálogo de productos, controlen las transacciones realizadas y obtengan información consolidada sobre el desempeño de las ventas.

La aplicación estará implementada bajo una **arquitectura en capas**, lo que permitirá una separación clara entre la interfaz de presentación, la lógica de negocio y el acceso a datos. De esta manera, el sistema será modular, mantenible y escalable, garantizando una estructura limpia que facilite futuras ampliaciones o modificaciones.

El desarrollo se realizará utilizando **Python** como lenguaje principal, junto con el framework **Flask** para la creación de la API RESTful, y **PostgreSQL** como sistema de gestión de base de datos relacional. Esta combinación de tecnologías ofrece un entorno robusto, flexible y ampliamente utilizado en el desarrollo de sistemas empresariales modernos.

## ****Objetivos del sistema****

* Desarrollar una aplicación funcional que permita la **gestión integral de ventas** y productos.
* Implementar una **arquitectura en capas** que promueva la mantenibilidad, escalabilidad y separación de responsabilidades.
* Exponer una **API RESTful** que permita la comunicación entre las capas y facilite la integración con futuras interfaces web o móviles.
* Garantizar la **persistencia y consistencia de los datos** mediante PostgreSQL y un ORM como SQLAlchemy.
* Sentar las bases para un sistema adaptable a futuras mejoras, como el manejo de usuarios y reportes avanzados.

## ****Motivación de la elección de la arquitectura****

La arquitectura en capas fue seleccionada por su capacidad para organizar el código de forma estructurada y separar las responsabilidades del sistema. En este enfoque, cada capa tiene un propósito bien definido:

* **Capa de Presentación:** Gestiona la interacción entre el usuario o cliente (API REST) y el sistema.
* **Capa de Lógica de Negocio:** Contiene las reglas que gobiernan las operaciones de ventas, cálculos y validaciones.
* **Capa de Acceso a Datos:** Gestiona la persistencia y recuperación de información desde la base de datos PostgreSQL.

Este modelo ofrece claridad en el desarrollo, facilidad en las pruebas y mayor control sobre la evolución del software, permitiendo que cada capa se pueda modificar sin afectar directamente a las demás. Además, la arquitectura en capas se ajusta de manera ideal al tipo de aplicación propuesto, dado que combina componentes de backend, procesamiento de negocio y almacenamiento de datos de forma coherente y escalable.

# Análisis de Requisitos

El análisis de requisitos constituye una etapa esencial en la definición del sistema, ya que permite identificar las necesidades de los usuarios, traducirlas en funcionalidades específicas y establecer las condiciones de calidad que el software debe cumplir. A continuación, se detallan las **Historias de Usuario (HU)**, los **Requisitos Funcionales (RF)** y los **Requisitos No Funcionales (RNF)** que guiarán el desarrollo del Sistema de Gestión de Ventas en Línea bajo arquitectura en capas.

## ****Historias de Usuario****

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU01 | TÍTULO: Registro de productos |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: registrar nuevos productos en el sistema  PARA: mantener actualizado el catálogo y facilitar la gestión del inventario. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se deben ingresar nombre, descripción, categoría, precio y stock.  - El sistema valida campos obligatorios y evita duplicados.  - Muestra confirmación visual y mensaje de éxito.  - Los productos se visualizan inmediatamente en el catálogo. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU02 | TÍTULO: Edición de productos |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: editar la información existente de los productos  PARA: corregir errores, actualizar precios o modificar detalles relevantes. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Solo los administradores pueden editar.  - Los cambios se guardan y reflejan en tiempo real.  - Se muestra mensaje de confirmación.  - No se permiten campos vacíos. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU03 | TÍTULO: Eliminación de productos |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: eliminar productos obsoletos o descontinuados  PARA: mantener la base de datos limpia y actualizada. | | |
| ESTIMACION: 7 | | PRIORIDAD: Media |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se solicita confirmación antes de eliminar.  - El producto se elimina lógicamente o físicamente según configuración.  - Se muestra mensaje de éxito.  - El catálogo se actualiza automáticamente. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU04 | TÍTULO: Visualización de productos |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: visualizar todos los productos disponibles  PARA: conocer las opciones que puedo comprar. | | |
| ESTIMACION: 9 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se listan productos con nombre, precio, stock y categoría.  - No se muestran productos sin stock.  - La visualización es paginada o con scroll.  - Diseño adaptable a distintos dispositivos. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU05 | TÍTULO: Búsqueda de productos |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: buscar productos por nombre o categoría  PARA: encontrar rápidamente lo que deseo adquirir. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - El buscador filtra resultados en tiempo real.  - Se puede buscar por coincidencias parciales.  - Resultados relevantes se muestran con imagen y precio.  - No se muestran productos inactivos. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU06 | TÍTULO: Carrito de compras |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: agregar, modificar o quitar productos de un carrito de compras  PARA: organizar mi pedido antes de realizar el pago. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se pueden añadir, modificar cantidad o eliminar productos del carrito.  - El carrito muestra subtotal y total actualizado.  - Persiste durante la sesión del usuario.  - No se permite agregar más unidades que el stock disponible. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU07 | TÍTULO: Registro de compras |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: registrar una compra seleccionando productos y cantidades específicas  PARA: finalizar mi pedido y obtener mi comprobante de pago. | | |
| ESTIMACION: 7 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - El sistema valida stock disponible antes de confirmar.  - Actualiza el inventario automáticamente.  - Genera comprobante con detalle de compra.  - La operación se guarda en el historial del cliente. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU08 | TÍTULO: Confirmación de compra |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: recibir una confirmación con el resumen de mi compra  PARA: verificar que mi pedido se registró correctamente. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se muestra número de transacción y resumen del pedido.  - El comprobante se puede descargar o enviar al correo.  - La confirmación aparece tras un pago exitoso.  - Incluye fecha, monto y lista de productos. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU09 | TÍTULO: Historial de compras del cliente |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: consultar mi historial de compras anteriores  PARA: revisar mis pedidos pasados y comprobar mis gastos. | | |
| ESTIMACION: 7 | | PRIORIDAD: Media |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se listan las compras ordenadas por fecha.  - Cada registro muestra número, monto, fecha y estado.  - Se puede filtrar por rango de fechas.  - Permite visualizar detalles individuales. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU10 | TÍTULO: Historial de ventas administrador |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: consultar el historial completo de ventas realizadas  PARA: analizar el rendimiento comercial y controlar las operaciones. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Permite filtrar por rango de fechas, cliente o producto.  - Se muestran totales de ventas y número de transacciones.  - Datos exportables a Excel o PDF.  - Solo accesible para administradores. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU11 | TÍTULO: Reporte de ventas |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: generar reportes de ventas detallados  PARA: apoyar la toma de decisiones estratégicas en la empresa. | | |
| ESTIMACION: 9 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Reportes generados por período, producto o categoría.  - Formatos de exportación: PDF o CSV.  - Incluye gráficos o totales resumidos.  - Guardado histórico de reportes generados. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU12 | TÍTULO: Ranking de productos más vendidos |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: visualizar el ranking de productos más vendidos  PARA: planificar los inventarios y mejorar la oferta comercial. | | |
| ESTIMACION: 7 | | PRIORIDAD: Media |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se muestra ranking ordenado por cantidad de ventas.  - Permite filtrar por mes o período.  - Muestra nombre, categoría y total vendido.  - Se actualiza automáticamente según nuevas ventas. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU13 | TÍTULO: Gestión de usuarios |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: gestionar los usuarios del sistema (crear, editar, eliminar)  PARA: mantener el control y la seguridad de los accesos. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Solo los administradores pueden acceder.  - Validación de campos obligatorios (nombre, correo, rol).  - Confirmación antes de eliminar usuario.  - Cambios reflejados en tiempo real. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU14 | TÍTULO: Asignación de roles |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: asignar roles de usuario (cliente o administrador)  PARA: garantizar que cada usuario tenga los permisos adecuados. | | |
| ESTIMACION: 9 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Cada usuario tiene un rol único.  - El sistema valida permisos antes de cada acción.  - Cambios de rol registrados en bitácora.  - Roles definidos al momento de registro. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU15 | TÍTULO: Autenticación de usuarios |
| ROL/USUARIO: Usuario | | |
| DESCRIPCION: | | |
| COMO: Usuario registrado  QUIERO: autenticarme mediante credenciales seguras  PARA: acceder a las funcionalidades que me corresponden. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - El sistema valida usuario y contraseña con seguridad.  - Contraseñas cifradas en base de datos.  - Muestra mensajes claros en caso de error.  - Cierre de sesión disponible. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU16 | TÍTULO: Registro de clientes |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: registrarme en la plataforma  PARA: poder acceder al catálogo y realizar compras en línea. | | |
| ESTIMACION: 8 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Campos obligatorios: nombre, correo, contraseña.  - Validación de formato de correo y contraseñas seguras.  - Confirmación visual del registro exitoso.  - El usuario registrado obtiene rol “cliente”. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU17 | TÍTULO: Actualización de perfil |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: actualizar mis datos personales y credenciales  PARA: mantener mi información de usuario siempre vigente. | | |
| ESTIMACION: 6 | | PRIORIDAD: Media |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Se permite modificar nombre, correo y contraseña.  - Validación de formato y campos requeridos.  - Confirmación de cambios.  - Cambios visibles inmediatamente en el perfil. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU18 | TÍTULO: Alerta de bajo stock |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: recibir alertas automáticas cuando un producto tenga bajo stock  PARA: reabastecer inventario y evitar faltantes. | | |
| ESTIMACION: 7 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Notificación automática cuando un producto baja del mínimo.  - Se muestra lista de productos críticos.  - Alertas visibles en panel principal.  - Posibilidad de exportar lista. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU19 | TÍTULO: Pago seguro |
| ROL/USUARIO: Cliente | | |
| DESCRIPCION: | | |
| COMO: Cliente  QUIERO: realizar pagos de manera segura mediante una pasarela o simulador de pago  PARA: finalizar mis compras con confianza. | | |
| ESTIMACION: 10 | | PRIORIDAD: Alta |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Validación de método de pago antes de confirmar.  - En caso de éxito, se actualiza stock y genera comprobante.  - Transacciones seguras mediante HTTPS.  - Se guarda número de operación en historial. | | |
| MOCKUP: | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| HISTORIA DE USUARIO | ID: HU20 | TÍTULO: Respaldo de base de datos |
| ROL/USUARIO: Administrador | | |
| DESCRIPCION: | | |
| COMO: Administrador  QUIERO: realizar respaldos periódicos de la base de datos  PARA: prevenir la pérdida de información ante fallos o errores. | | |
| ESTIMACION: 7 | | PRIORIDAD: Media |
| CRITERIOS DE ACEPTACIÓN: | | |
| - Permite respaldo manual y programado.  - Notificación de éxito o error del respaldo.  - Descarga del archivo generado.  - Registro de fecha y hora del respaldo. | | |
| MOCKUP: | | |
|  | | |

## ****Requisitos Funcionales****

|  |  |
| --- | --- |
| **Código** | **Requisito Funcional** |
| **RF-01** | Permitir el registro, edición y eliminación de productos en el catálogo. |
| **RF-02** | Permitir búsquedas y filtrado de productos por nombre o categoría. |
| **RF-03** | Implementar un carrito de compras temporal que almacene los ítems seleccionados. |
| **RF-04** | Registrar ventas completas con actualización automática del stock. |
| **RF-05** | Generar y mostrar comprobantes de venta con detalles del pedido. |
| **RF-06** | Permitir autenticación y registro de usuarios mediante credenciales seguras. |
| **RF-07** | Gestionar roles y permisos para usuarios del sistema. |
| **RF-08** | Permitir la visualización del historial de compras y ventas. |
| **RF-09** | Generar reportes analíticos (productos más vendidos, ingresos, etc.). |
| **RF-10** | Permitir exportar reportes a formatos estándar (PDF, CSV). |
| **RF-11** | Notificar alertas automáticas de bajo stock. |
| **RF-12** | Habilitar procesos de pago seguros con validación de transacciones. |
| **RF-13** | Permitir la gestión completa de usuarios (crear, editar, eliminar). |
| **RF-14** | Implementar copias de seguridad manuales o automáticas. |
| **RF-15** | Exponer todas las funcionalidades a través de una API RESTful documentada. |

### Modularización de Requisitos Funcionales

#### Gestión de Usuarios

|  |  |
| --- | --- |
| Código | Descripción |
| RF-06 | Permitir autenticación y registro de usuarios mediante credenciales seguras. |
| RF-07 | Gestionar roles y permisos para usuarios del sistema. |
| RF-13 | Permitir la gestión completa de usuarios (crear, editar, eliminar). |

#### Gestión de Productos

|  |  |
| --- | --- |
| Código | Descripción |
| RF-01 | Permitir el registro, edición y eliminación de productos en el catálogo. |
| RF-02 | Permitir búsquedas y filtrado de productos por nombre o categoría. |
| RF-04 | Registrar ventas completas con actualización automática del stock. |

#### Ventas o Facturación

|  |  |
| --- | --- |
| Código | Descripción |
| RF-03 | Implementar un carrito de compras temporal que almacene los ítems seleccionados. |
| RF-05 | Generar y mostrar comprobantes de venta con detalles del pedido. |
| RF-12 | Habilitar procesos de pago seguros con validación de transacciones. |
| RF-08 | Permitir la visualización del historial de compras y ventas. |

#### Reportes y Analíticas

|  |  |
| --- | --- |
| Código | Descripción |
| RF-09 | Generar reportes analíticos (productos más vendidos, ingresos, etc.). |
| RF-10 | Permitir exportar reportes a formatos estándar (PDF, CSV). |

#### Notificaciones

|  |  |
| --- | --- |
| Código | Descripción |
| RF-11 | Notificar alertas automáticas de bajo stock. |

#### Configuración

|  |  |
| --- | --- |
| Código | Descripción |
| RF-14 | Implementar copias de seguridad manuales o automáticas. |

#### Interfaces de Integración

|  |  |
| --- | --- |
| Código | Descripción |
| RF-15 | Exponer todas las funcionalidades a través de una API RESTful documentada. |

## ****Requisitos No Funcionales****

|  |  |
| --- | --- |
| **Código** | **Requisito No Funcional** |
| **RNF-01** | El tiempo de respuesta promedio debe ser menor a 2 segundos por solicitud. |
| **RNF-02** | Todas las comunicaciones deben usar protocolos seguros (HTTPS, JWT). |
| **RNF-03** | La arquitectura en capas debe asegurar alta mantenibilidad y modularidad. |
| **RNF-04** | El sistema debe ser escalable horizontal y verticalmente según la demanda. |
| **RNF-05** | El código fuente debe seguir estándares de estilo y documentación técnica (PEP 8, docstrings). |
| **RNF-06** | La base de datos PostgreSQL debe garantizar integridad referencial y consistencia transaccional. |
| **RNF-07** | El sistema debe mantener una disponibilidad mínima del 95 % durante las operaciones. |
| **RNF-08** | Los mensajes de error y confirmación deben ser claros, comprensibles y uniformes. |
| **RNF-09** | La API debe ser compatible con aplicaciones web o móviles futuras. |
| **RNF-10** | Se deben realizar copias de seguridad automáticas diarias del entorno de producción. |

### Categorización de Requisitos No Funcionales Según la ISO 25010

|  |  |  |
| --- | --- | --- |
| Característica ISO 25010 | Cant. | Requisitos No Funcionales |
| Eficiencia de Desempeño | 2 | **RNF-01**: El tiempo de respuesta promedio debe ser menor a 2 segundos por solicitud.  **RNF-04**: El sistema debe ser escalable horizontal y verticalmente según la demanda. |
| Compatibilidad | 1 | **RNF-09**: La API debe ser compatible con aplicaciones web o móviles futuras. |
| Usabilidad | 1 | **RNF-08**: Los mensajes de error y confirmación deben ser claros, comprensibles y uniformes. |
| Fiabilidad | 2 | **RNF-07:** El sistema debe mantener una disponibilidad mínima del 95 % durante las operaciones.  **RNF-06:** La base de datos PostgreSQL debe garantizar integridad referencial y consistencia transaccional. |
| Seguridad | 2 | **RNF-10:** Se deben realizar copias de seguridad automáticas diarias del entorno de producción.  **RNF-02:** Todas las comunicaciones deben usar protocolos seguros (HTTPS, JWT). |
| Mantenibilidad | 2 | **RNF-03:** La arquitectura en capas debe asegurar alta mantenibilidad y modularidad.  **RNF-05:** El código fuente debe seguir estándares de estilo y documentación técnica (PEP 8, docstrings). |

# Selección y Descripción de la Arquitectura

El sistema propuesto se desarrollará utilizando una **arquitectura en capas**, una de las estructuras más consolidadas y efectivas en el diseño de aplicaciones empresariales.  
Este enfoque permite separar las responsabilidades del software en diferentes niveles lógicos, logrando una solución modular, mantenible y escalable.

La elección de esta arquitectura responde tanto a necesidades técnicas como académicas, buscando aplicar principios de diseño limpio (Clean Architecture) y buenas prácticas de ingeniería de software.

## ****Descripción General de la Arquitectura en Capas****

La **arquitectura en capas** organiza la aplicación en diferentes niveles que interactúan entre sí de forma controlada.

Cada capa cumple un rol específico y se comunica únicamente con la capa inmediata superior o inferior, garantizando una estructura ordenada y desacoplada. Para el **Sistema de Gestión de Ventas en Línea**, se implementarán tres capas principales:

### ****Capa de Presentación (Front/Interface Layer)****

* Responsable de la **interacción con el usuario o cliente externo** (a través de una API REST).
* Se desarrollará utilizando **Flask**, aprovechando su capacidad para exponer endpoints HTTP (GET, POST, PUT, DELETE).
* Esta capa recibirá solicitudes, validará datos y los enviará a la capa de lógica de negocio.
* También devolverá las respuestas en formato **JSON**, facilitando la integración futura con una interfaz web o aplicación móvil.

### ****Capa de Lógica de Negocio (Business Logic Layer)****

* Contiene las **reglas de negocio**, cálculos, validaciones y flujo de procesos del sistema.
* Gestiona operaciones como el registro de ventas, control de stock, generación de reportes o verificación de usuarios.
* En esta capa se ubicarán los **servicios, controladores y validadores**, garantizando que la lógica esté centralizada y separada de la presentación.
* Su independencia permite realizar **pruebas unitarias** de manera sencilla.

### ****Capa de Acceso a Datos (Data Access Layer)****

* Gestiona la **persistencia y recuperación de información** desde la base de datos PostgreSQL.
* Utiliza **SQLAlchemy** como ORM (Object Relational Mapper) para mapear las entidades del sistema (Producto, Venta, Usuario, etc.) a tablas de la base de datos.
* Aísla los detalles técnicos del almacenamiento, de modo que la lógica de negocio no dependa directamente de SQL.
* Facilita la migración a otras bases de datos en el futuro, si fuese necesario.

## ****Flujo de Comunicación entre Capas****

El funcionamiento general del sistema seguirá el siguiente flujo:

1. El usuario (cliente o administrador) realiza una solicitud HTTP a través de la **API REST** (por ejemplo: registrar una venta).
2. La **capa de presentación** recibe la solicitud, valida los datos y los envía a la **capa de lógica de negocio**.
3. La **lógica de negocio** procesa las reglas correspondientes (verifica stock, calcula totales, guarda registros, etc.) y solicita la persistencia a la **capa de datos.**
4. La **capa de acceso a datos** ejecuta las operaciones necesarias sobre la base PostgreSQL y retorna el resultado a la lógica.
5. Finalmente, el sistema responde al usuario con un mensaje o resultado estructurado en formato JSON.

Este flujo garantiza **claridad, bajo acoplamiento y alto nivel de cohesión**, principios esenciales del desarrollo de software moderno.

## ****Justificación de la Elección****

La arquitectura en capas fue seleccionada por su simplicidad, claridad estructural y adaptabilidad al contexto del proyecto.

A continuación, se detallan las principales razones técnicas y de negocio que respaldan esta elección:

#### ****Razones Técnicas****

* **Separación de responsabilidades:** cada capa tiene funciones bien definidas, facilitando la localización y corrección de errores.
* **Facilidad de mantenimiento:** los cambios en la interfaz o la base de datos no afectan directamente la lógica de negocio.
* **Escalabilidad controlada:** permite añadir nuevas capas o servicios (como autenticación avanzada o analítica) sin reestructurar el sistema.
* **Pruebas unitarias eficientes:** la lógica puede probarse sin depender del entorno de base de datos o interfaz.
* **Compatibilidad con Flask:** el framework se adapta naturalmente a un diseño multicapa, permitiendo un enrutamiento limpio y modular.

#### ****Razones de Negocio y Académicas****

* **Adaptación a sistemas reales:** la mayoría de las aplicaciones empresariales medianas y grandes utilizan este tipo de arquitectura.
* **Facilidad de comprensión para equipos académicos:** su estructura por niveles facilita el aprendizaje y documentación.
* **Base para futuras mejoras:** el sistema puede evolucionar hacia arquitecturas más complejas (como microservicios o Clean Architecture) sin perder coherencia.
* **Reducción de riesgos:** al aislar componentes, se minimiza el impacto de los errores o cambios en una parte específica del sistema.

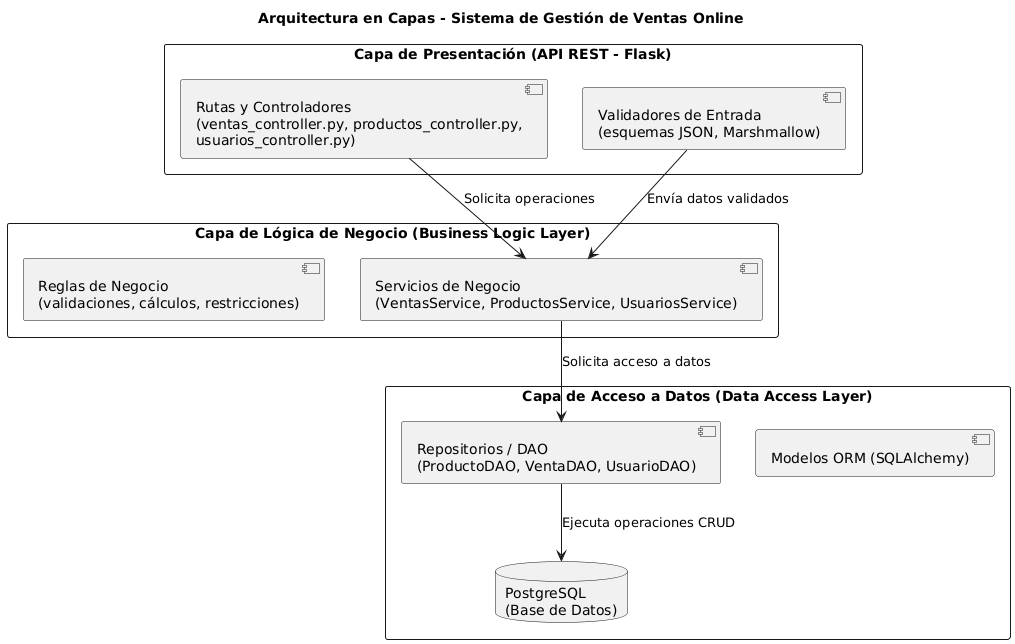
## ****Tecnologías y Herramientas Empleadas****

|  |  |  |
| --- | --- | --- |
| **Capa** | **Tecnología / Herramienta** | **Descripción** |
| Presentación | **Flask (Python)** | Framework ligero para desarrollo de APIs RESTful. |
| Lógica de Negocio | **Servicios y controladores Flask, Blueprints** | Manejo modular de rutas, validaciones y reglas. |
| Acceso a Datos | **SQLAlchemy + PostgreSQL** | ORM que abstrae la manipulación de datos con modelos relacionales. |
| Documentación | **Swagger / Postman** | Pruebas y documentación de endpoints. |
| Control de Versiones | **Git + GitHub** | Seguimiento del progreso del desarrollo. |
| Entorno de Pruebas | **pytest / unittest** | Validación de componentes y servicios. |

# Diseño Preliminar

El **diseño preliminar** traduce los requisitos del sistema y las decisiones arquitectónicas en una representación estructurada, donde se visualizan los principales módulos, clases y flujos de información.  
En esta etapa se definen los **componentes clave** y su interacción dentro de la **arquitectura en capas**, utilizando **diagramas UML** para representar el diseño de alto nivel.

## ****Diagrama General del Sistema (UML de Capas)****



## ****Diagrama de Componentes****

## ****Diagrama de Clases****

# Plan de Trabajo

El desarrollo del **Sistema de Gestión de Ventas en Línea**, basado en **arquitectura en capas**, se planifica siguiendo un enfoque incremental y estructurado.  
El plan contempla desde la fase de análisis hasta la integración y pruebas finales, asegurando la trazabilidad entre los requisitos, el diseño y la implementación.

## ****Cronograma de Actividades****

|  |  |  |  |
| --- | --- | --- | --- |
| **Fase** | **Descripción** | **Duración Estimada** | **Entregables / Resultados** |
| **F1. Análisis y levantamiento de requisitos** | Revisión de las HU, identificación de actores, definición de RF y RNF. | Semana 1 | Documento de requisitos, lista validada de HU, RF y RNF. |
| **F2. Diseño arquitectónico y modelo UML** | Definición de la arquitectura en capas, diagramas de clases, componentes y capas. | Semana 2 | Diseño UML aprobado y documentación arquitectónica. |
| **F3. Configuración del entorno y base de datos** | Instalación de Flask, SQLAlchemy y PostgreSQL. Creación del esquema inicial de base de datos. | Semana 3 | Estructura base del proyecto y BD inicial funcional. |
| **F4. Implementación de la capa de acceso a datos (DAL)** | Desarrollo de modelos ORM y repositorios para entidades principales. | Semana 4 | Código funcional de conexión y persistencia (SQLAlchemy). |
| **F5. Implementación de la lógica de negocio (BLL)** | Programación de los servicios de productos, ventas y usuarios. | Semana 5 | Módulos de negocio validados y probados unitariamente. |
| **F6. Desarrollo de la capa de presentación (API REST)** | Creación de endpoints Flask, validación de datos y controladores. | Semana 6 | Endpoints funcionales para CRUD de productos, usuarios y ventas. |
| **F7. Pruebas e integración** | Verificación integral de las tres capas y validación de casos de uso principales. | Semana 7 | Resultados de pruebas, corrección de errores, versión estable. |
| **F8. Documentación y presentación final** | Elaboración del informe final y presentación de resultados. | Semana 8 | Documento técnico completo y demo funcional. |

## ****Próximos Pasos****

* Para avanzar hacia la **versión final del proyecto**, se plantea el siguiente conjunto de actividades:
* **Implementar los módulos definidos en el diseño UML**, comenzando por el acceso a datos y luego la lógica de negocio.
* **Diseñar los endpoints de la API REST** siguiendo las convenciones HTTP (GET, POST, PUT, DELETE).
* **Configurar las validaciones y serializaciones** usando Marshmallow o pydantic.
* **Realizar pruebas unitarias e integración continua** para cada capa.
* **Implementar control de errores y seguridad básica** (manejo de tokens o autenticación JWT).
* **Documentar la API con Swagger o Postman,** facilitando su evaluación.
* **Evaluar mejoras visuales futuras**, como una interfaz web con Flask Templates o React.
* **Preparar la presentación y demostración final** con ejemplos de flujo de venta y generación de reportes.

# Conclusiones

El desarrollo del proyecto **“Sistema de Gestión de Ventas en Línea”,** bajo el enfoque de **Arquitectura en Capas**, ha permitido consolidar un marco técnico claro y ordenado para la construcción de aplicaciones empresariales escalables y mantenibles.

Durante el proceso, se realizó un análisis completo de los **requisitos funcionales y no funcionales**, complementado con **historias de usuario** que reflejan necesidades reales del negocio. A partir de ello, se diseñó una arquitectura sólida compuesta por tres capas bien diferenciadas (**Presentación, Lógica de Negocio y Acceso a Datos)**, garantizando así una separación adecuada de responsabilidades.

El diseño UML elaborado, que incluye los diagramas **de capas, componentes y clases,** proporciona una visión integral del sistema y establece las bases para una implementación coherente con las buenas prácticas de desarrollo en Python y Flask. Este diseño asegura que cada módulo del sistema pueda evolucionar de forma independiente sin comprometer la integridad general del software.

Desde una perspectiva académica, la experiencia refuerza la importancia de la arquitectura en capas como un modelo de referencia para el desarrollo estructurado. Su aplicación fomenta la mantenibilidad del código, facilita la detección de errores, y mejora la capacidad del sistema para adaptarse a nuevas funcionalidades.

Por tanto, el proyecto no solo cumple con los objetivos planteados, sino que también sienta las bases para fases futuras, como la integración de una interfaz gráfica de usuario o la evolución hacia arquitecturas más modernas como **Clean Architecture o microservicios**.

En conclusión, la aplicación del enfoque por capas demuestra ser una estrategia efectiva para alcanzar un software **robusto, modular y de alta calidad**, alineado con los estándares profesionales y educativos del desarrollo de sistemas.