Proyecto Certificador de Desarrollo de Software 1

Tema Nº11:

Fase de Implementación – Sprint 3

Indicador de logro Nº11:Desarrolla los entregables del producto de software en el Sprint 3, aplicando la Fase de Implementación del Framework Scrum.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº11:**

FASE DE IMPLEMENTACIÓN – SPRINT 3

Al llegar al SPRINT 3 del Scrum, nos embestimos con la implementación del proyecto. Es decir, la sala de reuniones donde se discute el sprint y se explora cómo optimizar el trabajo de cada grupo Scrum para darle forma definitiva al proyecto.

En la implementación se cumple con los siguientes procesos:

1. Crear entregables.
2. Realizar *daily stand-up.*
3. Refinanciamiento del backlog priorizado del producto.

En la fase de implementación o desarrollo no deberían hacerse cambios innecesarios de última hora (se supone que para evitarlo existe una fase de planificación). Aún así, si necesitas hacer un movimiento que será clave para el éxito del sprint, no dudes en proceder. Eso sí: discútelo primero con el grupo y los stakeholders, para que no haya confusiones o malos entendidos.

**Reuniones Scrum Diarias o Daily Meetings**

Cada persona debe de responder las siguientes preguntas:

* ¿Qué hiciste ayer?
* ¿En qué trabajarás hoy?
* ¿Qué obstáculos han surgido?

**Ceremonia: Sprint Review**

* El equipo de desarrollo comenta posteriormente qué ha ocurrido durante el Sprint 1 y 2 , los impedimentos que se han encontrado, así como soluciones tomadas y actualizan a los stakeholders con la situación del equipo.

Ceremonia: Reunión diaria (Daily Scrum Meeting).

### 3ª ceremonia: Sprint Review

El Sprint Review es la reunión que ocurre al final del Sprint, generalmente el último viernes del Sprint, donde el product owner y el Develpment Team presentan a los stakeholders el incremento terminado para su inspección y adaptación correspondientes. En esta reunión organizada por el product owner se estudia cuál es la situación y se actualiza el Product Backlog con las nuevas condiciones que puedan afectar al negocio.

Por un lado, se revisará el incremento terminado. Se mostrará el software funcionando en producción y los stakeholders tendrán la oportunidad de hacer cuantas preguntas estimen oportunas sobre el mismo. El software funcionando ha sido validado previamente por el product owner, que se ha encargado de trabajar con el equipo durante el Sprint para asegurarse que cumple con la Definition of Done y, efectivamente, hace que el Sprint Goal sea válido. Si no existe software funcionando, el Sprint Review carece de sentido, aunque en ciertas ocasiones y oportunidades se sigue manteniendo.

El Development Team tiene que tener un papel importante en esta reunión. Muchas veces no es el product owner quien demuestra el incremento producido, sino que son los propios miembros del Development Team quienes lo hacen. Es una buena práctica no sólo el que lo lleven a cabo, sino también el que lo hagan de forma rotatoria y, tras varios Sprints, hayan participado todos.

El equipo de desarrollo comenta posteriormente qué ha ocurrido durante el Sprint, los impedimentos que se han encontrado, así como soluciones tomadas y actualizan a los stakeholders con la situación del equipo. Por último, el product owner actualiza -con la información de negocio recibida en esta reunión- el Product Backlog para el siguiente Sprint.

En contra de lo que comúnmente se cree, el Sprint Review no se trata de una demo para un cliente o para los stakeholders o incluso para el product owner, ni es tampoco una reunión para felicitar al Equipo de Desarrollo. Es una reunión de trabajo, una de las más importantes porque sirve para marcar la estrategia de negocio. La duración estimada en el estándar para un Sprint Review es de 8 horas para un Sprint de 4 semanas, aunque habitualmente estas reuniones se ejecutan en un entorno de entre 2 y 3 horas.

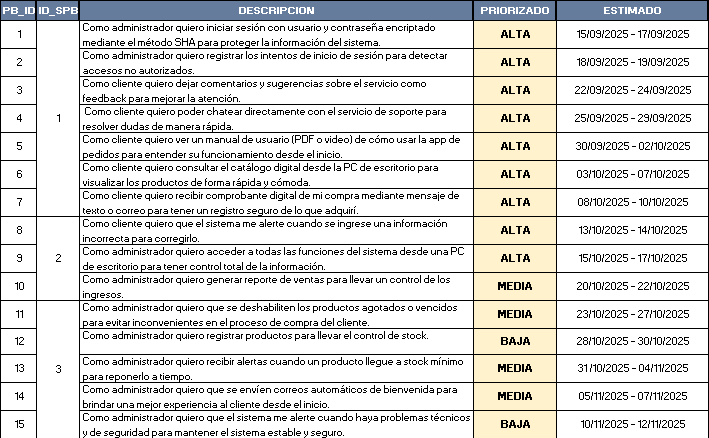
**Entregables del producto**

* + Creación de plantilla de todos los artefactos Scrum.
  + Uso de la arquitectura y patrones de diseño de software.
  + Uso de buenas prácticas de análisis y diseño de desarrollo de software.
  + Uso de buenas prácticas de codificación del software.
  + Atributos de calidad de software.
  + Integración de todos los artefactos Scrum en una sola plantilla Scrum

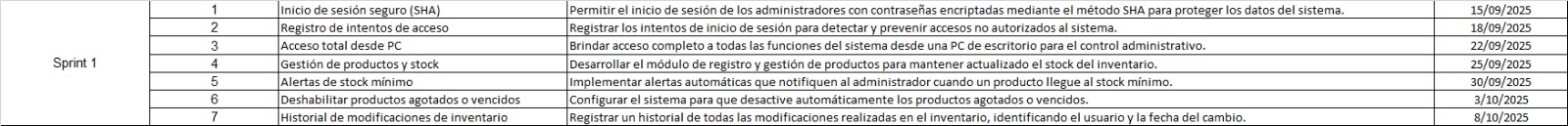
Desarrollo de la Guia\_Artefactos Scrum\_09

* Caso: Ventas
* Bodega: “WIVAGU”

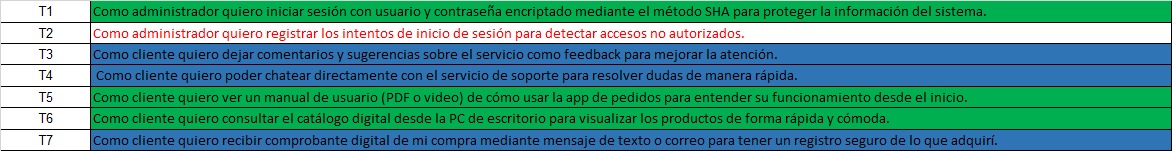
Product Backlog Priorizado

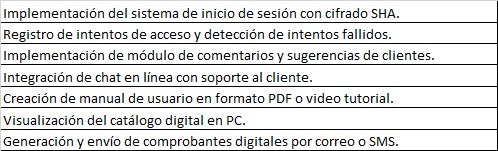


DETERMINACIÓN DE LA LONGITUD DEL SPRINT 1



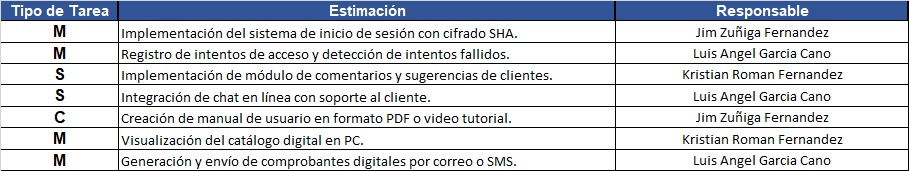
Desarrollo del Sprint 1





Ceremonias Daily Scrum Meeting-Sprint 1





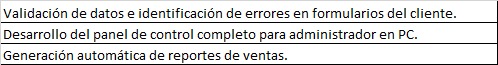


DETERMINACIÓN DE LA LONGITUD DEL SPRINT 2



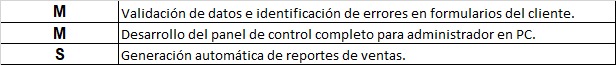
Desarrollo del Sprint 2

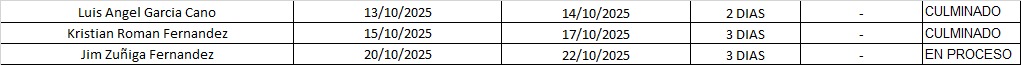




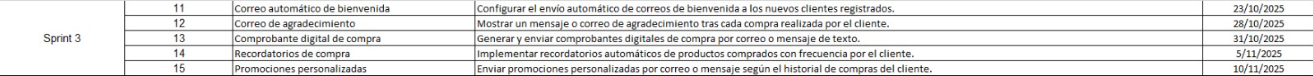
Ceremonias Daily Scrum Meeting-Sprint 2





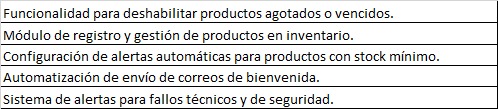
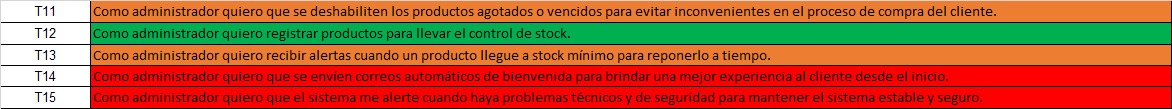


DETERMINACIÓN DE LA LONGITUD DEL SPRINT 3

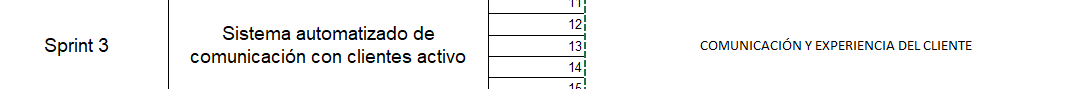
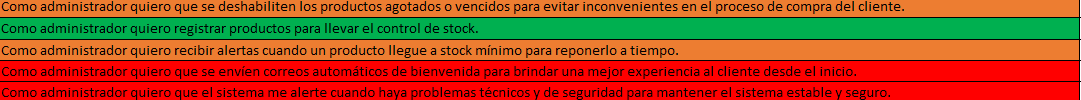
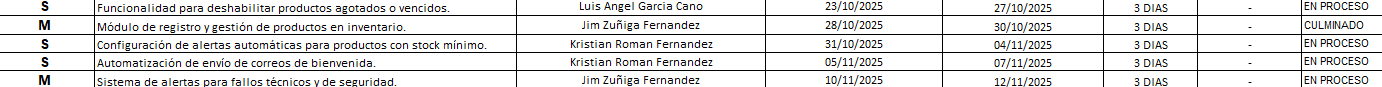


Sprint Backlog

Desarrollo del Sprint 3

****

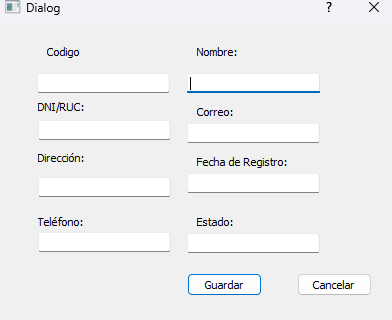
Ceremonias Daily Scrum Meeting-Sprint 3

****

Desarrollo de un CRUD- Tabla CLIENTES

SqlServer & Python

* Gestión de Clientes





Descripción del módulo u interfaz.

**Este módulo sirve para que el Administrador puede buscar y agregar nuevos clientes con nombre, DNI, dirección, correo y teléfono.**

**Se pueden modificar o eliminar registros.**

def abrir\_consulta\_cliente(self):

        self.consulta = uic.loadUi("Ui/CONSULTA\_CLIENTE.ui")

        self.consulta.show()

        self.menu.close()

        self.consulta.btnRegresarCliente.clicked.connect(self.regresar\_menu\_desde\_consulta)

        self.consulta.btnAgregarCliente.clicked.connect(self.abrir\_form\_cliente)

    def abrir\_form\_cliente(self):

        self.form\_cliente = uic.loadUi("Ui/FORM\_CLIENTE.ui")

        self.form\_cliente.show()

        self.form\_cliente.btnGuardarCliente.clicked.connect(self.guardar\_cliente\_desde\_form)

        self.form\_cliente.btnCancelar.clicked.connect(self.cancelar\_form\_cliente)

    def guardar\_cliente\_desde\_form(self):

        # Obtener los datos del form

        codigo = self.form\_cliente.txtCodigo.text()

        nombre = self.form\_cliente.txtNombre.text()

        dni = self.form\_cliente.txtDNI.text()

        direccion = self.form\_cliente.txtDireccion.text()

        telefono = self.form\_cliente.txtTelefono.text()

        correo = self.form\_cliente.txtCorreo.text()

        fecha\_registro = self.form\_cliente.txtFecha.text()

        estado = self.form\_cliente.txtEstado.text()

        # Validar campos obligatorios

        if codigo == "" or nombre == "":

            QMessageBox.warning(self.form\_cliente, "Error", "Debe ingresar al menos código y nombre.")

            return

        # Agregar a la tabla

        fila = self.consulta.tblClientes.rowCount()

        self.consulta.tblClientes.insertRow(fila)

        self.consulta.tblClientes.setItem(fila, 0, QTableWidgetItem(codigo))

        self.consulta.tblClientes.setItem(fila, 1, QTableWidgetItem(nombre))

        self.consulta.tblClientes.setItem(fila, 2, QTableWidgetItem(dni))

        self.consulta.tblClientes.setItem(fila, 3, QTableWidgetItem(direccion))

        self.consulta.tblClientes.setItem(fila, 4, QTableWidgetItem(telefono))

        self.consulta.tblClientes.setItem(fila, 5, QTableWidgetItem(correo))

        self.consulta.tblClientes.setItem(fila, 6, QTableWidgetItem(fecha\_registro))

        self.consulta.tblClientes.setItem(fila, 7, QTableWidgetItem(estado))

        # Limpiar el formulario

        self.form\_cliente.txtNombre.clear()

        self.form\_cliente.txtDNI.clear()

        self.form\_cliente.txtDireccion.clear()

        self.form\_cliente.txtTelefono.clear()

        self.form\_cliente.txtCorreo.clear()

        self.form\_cliente.txtFecha.clear()

        self.form\_cliente.txtEstado.clear()

        QMessageBox.information(self.form\_cliente, "Éxito", "Cliente agregado correctamente.")

    def cancelar\_form\_cliente(self):

        self.form\_cliente.close()     # Cierra el formulario cliente

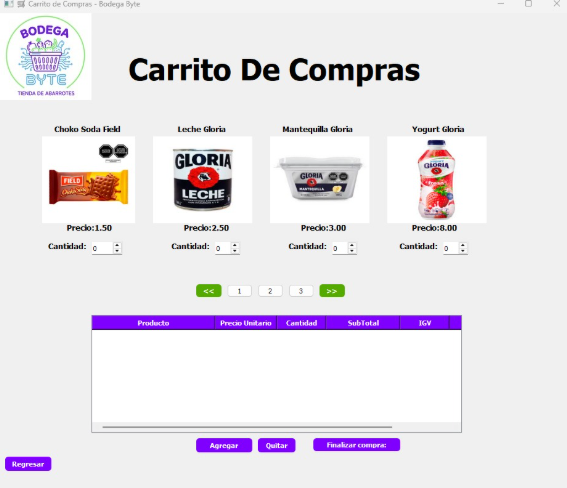
        self.consulta.show()          # Vuelve a mostrar la tabla de clientes

    def regresar\_menu\_desde\_consulta(self):

        self.consulta.close()

        self.abrir\_menu\_sistema()

* Carrito de compras



Descripción del módulo u interfaz:  
  
**Este módulo sirve para que cliente o el sistema muestra un catálogo visual de productos con su imagen referencial, nombre y precio unitario.**

**El usuario selecciona la cantidad deseada mediante controles numéricos y utiliza el botón "Agregar" para llevar los productos a la lista de pedido.**

**Se visualiza una tabla resumen (detalle de venta) que calcula en tiempo real el Precio Unitario, Cantidad, SubTotal e IGV por ítem.**

**Incluye controles de navegación (paginación) para ver más productos y botones para "Quitar" ítems o "Finalizar compra" para cerrar la transacción.**

import sys

import os

from PyQt5 import uic

from PyQt5.QtWidgets import (

QApplication, QMainWindow, QLabel, QTableWidgetItem, QMessageBox

)

from PyQt5.QtGui import QPixmap

class CarritoComprasApp(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

uic.loadUi("Ui/carritoCompras.ui", self)

self.setWindowTitle("🛒 Carrito de Compras - Bodega Byte")

# 📂 Ruta de las imágenes

self.ruta\_imagenes = os.path.join(os.path.dirname(\_\_file\_\_), "Imagenes")

# BOTÓN REGRESAR

self.btnRegresar.clicked.connect(self.Regresar)

# Navegación

self.btn1.clicked.connect(lambda: self.cambiarPagina(0))

self.btn2.clicked.connect(lambda: self.cambiarPagina(1))

self.btn3.clicked.connect(lambda: self.cambiarPagina(2))

self.btnAnterior.clicked.connect(self.paginaAnterior)

self.btnSiguiente.clicked.connect(self.paginaSiguiente)

# Botones carrito

self.btnAgregar.clicked.connect(self.agregarProducto)

self.btnQuitar.clicked.connect(self.quitarProducto)

self.btnFC.clicked.connect(self.finalizarCompra)

# =============================

# CONFIGURAR TABLA tblValores

# =============================

self.tblValores.setColumnCount(6)

self.tblValores.setHorizontalHeaderLabels([

"Producto", "Precio Unitario", "Cantidad",

"SubTotal", "IGV", "Total Pagar"

])

self.tblValores.setColumnWidth(0, 200)

self.tblValores.setColumnWidth(1, 100)

self.tblValores.setColumnWidth(2, 80)

self.tblValores.setColumnWidth(3, 120)

self.tblValores.setColumnWidth(4, 80)

self.tblValores.setColumnWidth(5, 120)

# ===== Datos de productos =====

self.paginas = [

[

("ChokoSoda Field", 1.50, "img05", "sp05"),

("Leche Gloria", 2.50, "img06", "sp06"),

("Mantequilla Gloria", 3.00, "img07", "sp07"),

("Yogurt Gloria", 8.00, "img08", "sp08"),

],

[

("Pan Bimbo Molde", 5.00, "img09", "sp09"),

("Inka Kola 3L", 12.00, "img10", "sp10"),

("Aceite Cocinero", 7.00, "img11", "sp11"),

("Coka Kola 3L", 11.50, "img12", "sp12"),

],

[

("Leche Entera Laive", 5.00, "img01", "sp01"),

("Chips Ahoy", 2.00, "img02", "sp02"),

("Oreos Mini", 3.50, "img03", "sp03"),

("Chocolate Sublime", 2.00, "img04", "sp04"),

],

]

# Página inicial

self.stackedProductos.setCurrentIndex(0)

self.cargarImagenes(0)

# =============================

# Navegación

# =============================

def cambiarPagina(self, index):

self.stackedProductos.setCurrentIndex(index)

self.cargarImagenes(index)

def paginaAnterior(self):

index = self.stackedProductos.currentIndex()

if index > 0:

self.cambiarPagina(index - 1)

def paginaSiguiente(self):

index = self.stackedProductos.currentIndex()

if index < self.stackedProductos.count() - 1:

self.cambiarPagina(index + 1)

# =============================

# Cargar imágenes

# =============================

def cargarImagenes(self, index):

grupos = [

["img05.jpg", "img06.jpg", "img07.jpg", "img08.jpg"],

["img09.jpg", "img10.jpg", "img11.jpg", "img12.jpg"],

["img01.jpg", "img02.jpg", "img03.jpg", "img04.jpg"],

]

pagina = self.stackedProductos.widget(index)

for archivo in grupos[index]:

label\_name = archivo.split(".")[0]

label = pagina.findChild(QLabel, label\_name)

if label:

ruta = os.path.join(self.ruta\_imagenes, archivo)

if os.path.exists(ruta):

pixmap = QPixmap(ruta)

label.setPixmap(pixmap)

label.setScaledContents(True)

# =============================

# AGREGAR PRODUCTOS

# =============================

def agregarProducto(self):

pagina\_actual = self.stackedProductos.currentIndex()

productos = self.paginas[pagina\_actual]

agregado = False

for nombre, precio, \_, spin\_name in productos:

spin = getattr(self, spin\_name)

cantidad = spin.value()

if cantidad > 0:

subtotal = precio \* cantidad

igv = subtotal \* 0.18

total\_pagar = subtotal + igv

fila = self.tblValores.rowCount()

self.tblValores.insertRow(fila)

self.tblValores.setItem(fila, 0, QTableWidgetItem(nombre))

self.tblValores.setItem(fila, 1, QTableWidgetItem(f"S/. {precio:.2f}"))

self.tblValores.setItem(fila, 2, QTableWidgetItem(str(cantidad)))

self.tblValores.setItem(fila, 3, QTableWidgetItem(f"S/. {subtotal:.2f}"))

self.tblValores.setItem(fila, 4, QTableWidgetItem(f"S/. {igv:.2f}"))

self.tblValores.setItem(fila, 5, QTableWidgetItem(f"S/. {total\_pagar:.2f}"))

spin.setValue(0)

agregado = True

if not agregado:

QMessageBox.warning(self, "Aviso", "Seleccione al menos un producto.")

# =============================

# QUITAR PRODUCTO

# =============================

def quitarProducto(self):

fila = self.tblValores.currentRow()

if fila >= 0:

self.tblValores.removeRow(fila)

else:

QMessageBox.warning(self, "Error", "Seleccione un producto para quitar.")

# =============================

# FINALIZAR COMPRA

# =============================

def finalizarCompra(self):

if self.tblValores.rowCount() == 0:

QMessageBox.warning(self, "Vacío", "No hay productos en el carrito.")

return

QMessageBox.information(self, "Compra finalizada", "Gracias por su compra.")

self.tblValores.setRowCount(0)

# =============================

# REGRESAR

# =============================

def Regresar(self):

from Vista.CuadroNo import CuadroNoApp

self.ventanaNoCliente = CuadroNoApp()

self.ventanaNoCliente.show()

self.hide()

if \_\_name\_\_ == "\_\_main\_\_":

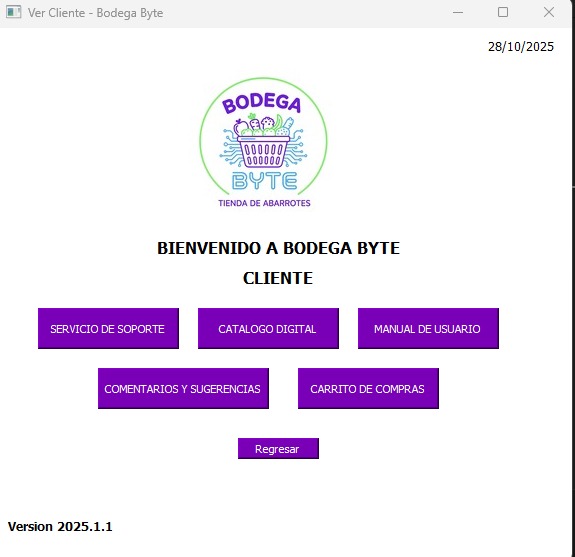
app = QApplication(sys.argv)

ventana = CarritoComprasApp()

ventana.show()

sys.exit(app.exec\_())

* Interfaz principal para el cliente



Descripción del módulo e interfaz:

**Este módulo sirve par a que el cliente explore entre nuestras distintas y diversas maneras de comunicarnos eh interactuar con el mediaste módulos ajustados a su comodidad y así pueda familiarizarse con ellos.**

from PyQt5 import uic

from PyQt5.QtWidgets import QMainWindow

from PyQt5.QtGui import QDesktopServices

from PyQt5.QtCore import QUrl

class verClienteApp(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

# Cargamos el diseño directamente desde tu carpeta Ui

uic.loadUi("Ui/vercliente.ui", self)

# Título de la ventana

self.setWindowTitle("Ver Cliente - Bodega Byte")

# Conectamos los botones a sus funciones

self.btnRegresar.clicked.connect(self.volver)

# (Opcional) Conexión de otros botones del menú

self.btnSoporte.clicked.connect(self.soporte)

self.btnCatalogo.clicked.connect(self.catalogo)

self.btnManual.clicked.connect(self.manual)

self.btnCyS.clicked.connect(self.comentarios)

self.btnCarrito.clicked.connect(self.carrito)

# -----------------------------

# FUNCIONES DE LOS BOTONES

# -----------------------------

def volver(self):

from Vista.clienteNo import ClienteNoApp

self.ventanaPrincipal = ClienteNoApp()

self.ventanaPrincipal.show()

self.close()

def soporte(self):

numero = "51900499583"

mensaje = "Hola, necesito soporte con Bodega Byte."

url = f"https://wa.me/{numero}?text={mensaje.replace(' ', '%20')}"

QDesktopServices.openUrl(QUrl(url))

def catalogo(self):

# Enlace público directo al PDF en Google Drive

ruta\_pdf = "https://drive.google.com/file/d/1KraI0pksaovDd8a\_5MLmQg8y5HsCTE73/view?usp=sharing"

QDesktopServices.openUrl(QUrl(ruta\_pdf))

def manual(self):

ruta\_manual = "https://drive.google.com/file/d/1hBLB72Ex2yAQY2wMIdJ-rQVkxVvYzmf0/view?usp=sharing" # 🔁 Cambia esta ruta si deseas

QDesktopServices.openUrl(QUrl(ruta\_manual))

def comentarios(self):

numero = "51993431336"

mensaje = "Hola, quiero brindar una sugerencia para mejorar el servicio o la aplicación Bodega Byte."

url = f"https://wa.me/{numero}?text={mensaje.replace(' ', '%20')}"

QDesktopServices.openUrl(QUrl(url))

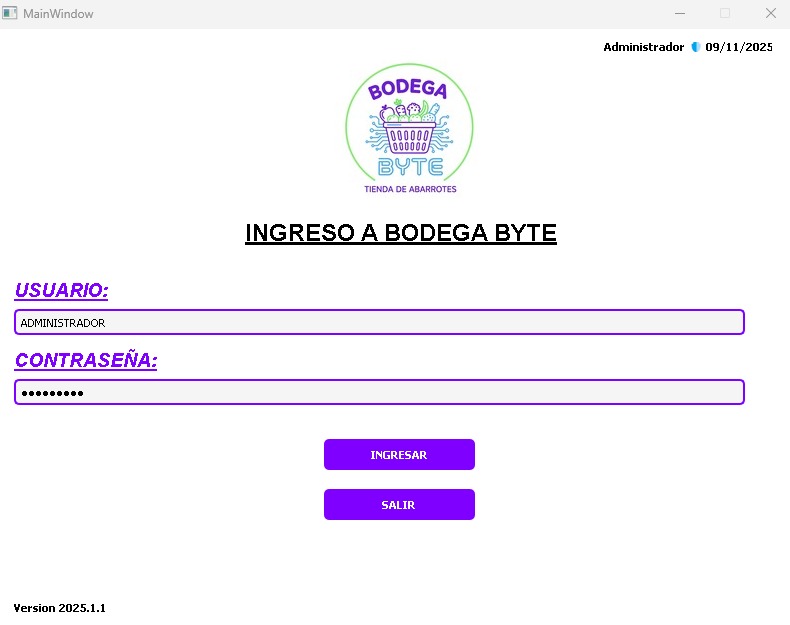
def carrito(self):

from Vista.carrito import CarritoComprasApp

self.ventanaCarrito = CarritoComprasApp()

self.ventanaCarrito.show()

self.close()

* Ingreso al sistema

Descripción del módulo e interfaz:

Este módulo sirve para que el administrador pueda ingresar de manera segura con su correo y contraseña en los campos correspondientes.

El sistema valida la información y permite el acceso al menú principal según el rol asignado.

Si el ingreso falla, el intento se registra para monitorear accesos no autorizados.

# ===================== LOGIN =====================

def validar\_login(self):

usuario = self.login.txtUsuario.text()

contrasena = self.login.txtContrasena.text()

hash\_contra = encriptar\_sha256(contrasena)

sha\_valida = encriptar\_sha256("1234")

if usuario == "ADMINISTRADOR" and hash\_contra == sha\_valida:

QMessageBox.information(self, "Acceso permitido", "Bienvenido al sistema, Administrador.")

self.abrir\_menu\_sistema()

else:

QMessageBox.warning(self, "Acceso denegado", "Usuario o contraseña incorrectos.")

Modulo de inventario





****

Descripción del módulo e interfaz:

El módulo de inventario permite acceder a dos funciones principales.

Registro de Inventario: donde se registran las salidas o entradas de productos.

Lista de Inventario: donde se visualizan los productos, su stock actual, mínimos y alertas.

El menú ofrece un entorno simple para navegar entre ambas funciones, permitiendo mantener un control rápido y ordenado del stock general de la tienda.

# ----------------- LISTA DE INVENTARIO -----------------

def abrir\_lista\_inventario(self):

self.lista\_inventario = uic.loadUi("Ui/LISTA\_INVENTARIO.ui")

self.lista\_inventario.show()

self.inventario.close()

self.lista\_inventario.btnRegresarLista.clicked.connect(self.regresar\_inventario\_desde\_lista)

def regresar\_inventario\_desde\_lista(self):

self.lista\_inventario.close()

self.abrir\_inventario()