

Universidad Autónoma De Tamaulipas

Facultad de Ingeniería Tampico

Ingeniería En Sistemas Computacionales

Código con Pantallas

Materia: Programación Avanzada

Alumna: Castillo Aviles Anahi

Matricula: a2233336121

Profesor: Ing. Alvarez Navarro Eduardo

Fecha:

Semestre: 3°

Grupo: “I”

Contenido

[Login 3](#_Toc198568309)

[Menú principal 21](#_Toc198568310)

[Venta 36](#_Toc198568311)

[Reportes 93](#_Toc198568312)

[Inventario 304](#_Toc198568313)

[Cliente 465](#_Toc198568314)

[Proveedores 541](#_Toc198568315)

[Usuario 576](#_Toc198568316)

# Login

package Main;

import Controlador.ventanas;

import Vista.reportes;

import Vista.Login;

import Vista.clientes;

import Vista.menuprincipal;

import Modelo.ClienteDAO;

import Modelo.Reportes;

import java.sql.Connection;

import javax.swing.SwingUtilities;

import ConexionBD.ConexionAccess;

import Controlador.ClientesContro;

import Controlador.ReportesControlador;

public class main {

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

Login login = new Login();

login.setVisible(true);

Connection conn = ConexionAccess.conectar();

});

}

}

package Modelo;

public class Usuario {

private String username;

private String password;

private String rol; // Para manejar roles como "Admin", "Cliente", etc.

public Usuario(String username, String password, String rol) {

this.username = username;

this.password = password;

this.rol = rol;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getRol() {

return rol;

}

public void setRol(String rol) {

this.rol = rol;

}

public boolean esAdministrador() {

// **TODO** Auto-generated method stub

if (rol.equalsIgnoreCase("Admin")) {

return true;

}

return false;

}

}

package ConexionBD;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.Properties;

import java.util.logging.Level;

import java.util.logging.Logger;

public class ConexionAccess {

private static final Logger logger = Logger.getLogger(ConexionAccess.class.getName());

private static final String URL = "jdbc:ucanaccess://C:\\Users\\Anahi\\eclipse-workspace\\punto\_venta\_2\\Punto\_Venta.accdb";

private static volatile Connection conn = null;

static {

try {

Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");

logger.info("Driver UCanAccess registrado correctamente");

} catch (ClassNotFoundException e) {

logger.log(Level.SEVERE, "Error al registrar el driver UCanAccess", e);

throw new ExceptionInInitializerError("No se pudo registrar el driver JDBC");

}

}

public static Connection conectar() {

try {

if (conn == null || conn.isClosed()) {

synchronized (ConexionAccess.class) {

if (conn == null || conn.isClosed()) {

Properties props = new Properties();

props.put("user", "");

props.put("password", "");

// Configuración simplificada (removemos JackcessOpener)

props.put("shutdown", "true");

conn = DriverManager.getConnection(URL, props);

logger.info("✅ Conexión a Access establecida exitosamente");

}

}

}

return conn;

} catch (SQLException e) {

logger.log(Level.SEVERE, "❌ Error al conectar con Access", e);

throw new RuntimeException("Error de conexión a la base de datos", e);

}

}

public static void cerrarConexion() {

if (conn != null) {

try {

// Intentar cerrar correctamente

if (!conn.isClosed()) {

conn.close();

logger.info("🔌 Conexión cerrada correctamente");

}

} catch (SQLException e) {

logger.log(Level.WARNING, "⚠️ Error al cerrar la conexión", e);

} finally {

conn = null; // Asegurar que se elimina la referencia

}

}

}

// Método para verificar el estado de la conexión

public static boolean verificarConexion() {

try {

if (conn == null || conn.isClosed()) {

return false;

}

// Prueba simple para verificar que la conexión funciona

return conn.createStatement().execute("SELECT 1 FROM MSysObjects WHERE 1=0");

} catch (SQLException e) {

logger.log(Level.WARNING, "⚠️ Error al verificar la conexión", e);

return false;

}

}

// Método para reconexión automática

public static Connection reconectar() {

cerrarConexion();

return conectar();

}

}

package ConexionBD;

import java.sql.\*;

public class TestConexion {

public static void main(String[] args) {

try (Connection conn = ConexionAccess.*conectar*()) {

System.***out***.println("✅ Conexión exitosa a Access!");

// Consultar clientes

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM clientes");

while (rs.next()) {

System.***out***.println(

"Cliente: " + rs.getString("nombre") +

" | Tel: " + rs.getString("telefono") +

" | Puntos: " + rs.getInt("puntos")

);

}

} catch (SQLException e) {

System.***err***.println("❌ Error en la conexión: " + e.getMessage());

}

}

}

package Vista;

import Modelo.Usuario;

import Modelo.UsuarioDAO;

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import Controlador.ventanas;

import java.awt.\*;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

public class Login extends JFrame {

private JTextField usernameField;

private JPasswordField passwordField;

private JButton loginButton;

private UsuarioDAO usuarioDAO;

public Login() {

configurarVentana();

initComponents();

usuarioDAO = new UsuarioDAO();

}

private void configurarVentana() {

setTitle("El Habanerito - Inicio de Sesión");

setSize(796, 513);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setResizable(false);

}

private void initComponents() {

JLayeredPane mainLayeredPane = new JLayeredPane();

mainLayeredPane.setPreferredSize(new Dimension(787, 455));

// Panel de fondo con gradiente

JPanel backgroundPanel = new JPanel() {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

GradientPaint gradient = new GradientPaint(0, 0, new Color(245, 245, 245),

0, getHeight(), new Color(230, 230, 230));

g2d.setPaint(gradient);

g2d.fillRect(0, 0, getWidth(), getHeight());

}

};

backgroundPanel.setBounds(0, 0, 787, 455);

mainLayeredPane.add(backgroundPanel, JLayeredPane.DEFAULT\_LAYER);

// Círculos decorativos

JPanel circleTopLeft = createCirclePanel(new Color(235, 215, 107, 150));

circleTopLeft.setBounds(-100, -100, 255, 255);

JPanel circleTopRight = createCirclePanel(new Color(216, 237, 88, 150));

circleTopRight.setBounds(787-120, -100, 255, 255);

JPanel circleBottomLeft = createCirclePanel(new Color(235, 184, 35, 150));

circleBottomLeft.setBounds(-100, 455-120, 255, 255);

JPanel circleBottomRight = createCirclePanel(new Color(241, 81, 17, 150));

circleBottomRight.setBounds(787-120, 455-120, 255, 255);

mainLayeredPane.add(circleTopLeft, JLayeredPane.PALETTE\_LAYER);

mainLayeredPane.add(circleTopRight, JLayeredPane.PALETTE\_LAYER);

mainLayeredPane.add(circleBottomLeft, JLayeredPane.PALETTE\_LAYER);

mainLayeredPane.add(circleBottomRight, JLayeredPane.PALETTE\_LAYER);

// Panel de contenido

JPanel contentPanel = new JPanel();

contentPanel.setOpaque(false);

contentPanel.setLayout(new BorderLayout());

contentPanel.setBounds(0, 0, 787, 455);

// Panel de título con logo y nombre

JPanel titlePanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 0));

titlePanel.setOpaque(false);

titlePanel.setBorder(BorderFactory.createEmptyBorder(30, 30, 30, 0));

// Logo clickeable con área oculta para registro

try {

ImageIcon originalIcon = new ImageIcon("imagen\\logo.png");

Image resizedImage = originalIcon.getImage().getScaledInstance(150, 150, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage)) {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

// Dibujar un pequeño indicador (solo para desarrollo, quitar en producción)

Graphics2D g2d = (Graphics2D) g.create();

g2d.setColor(new Color(255, 0, 0, 50));

g2d.fillRect(getWidth() - 30, getHeight() - 30, 20, 20);

g2d.dispose();

}

};

logo.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

logo.addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

// Área oculta: esquina inferior derecha (30x30px)

if (e.getX() > logo.getWidth() - 30 && e.getY() > logo.getHeight() - 30) {

mostrarRegistroOculto();

}

}

});

titlePanel.add(logo);

} catch (Exception e) {

System.err.println("Error cargando el logo: " + e.getMessage());

}

JLabel titleLabel = new JLabel("El Habanerito");

titleLabel.setFont(new Font("Arial", Font.BOLD, 28));

titleLabel.setForeground(new Color(70, 70, 70));

titlePanel.add(titleLabel);

contentPanel.add(titlePanel, BorderLayout.NORTH);

// Panel de formulario

JPanel formPanel = new JPanel();

formPanel.setOpaque(false);

formPanel.setLayout(new GridLayout(2, 2, 15, 15));

formPanel.setBorder(BorderFactory.createEmptyBorder(0, 150, 0, 150));

// Campos de texto

JLabel userLabel = new JLabel("Ingrese Usuario:");

userLabel.setFont(new Font("Arial", Font.BOLD, 14));

usernameField = new JTextField();

styleTextField(usernameField);

JLabel passLabel = new JLabel("Ingrese Contraseña:");

passLabel.setFont(new Font("Arial", Font.BOLD, 14));

passwordField = new JPasswordField();

passwordField.setEchoChar('•');

styleTextField(passwordField);

formPanel.add(userLabel);

formPanel.add(usernameField);

formPanel.add(passLabel);

formPanel.add(passwordField);

JPanel formContainer = new JPanel(new BorderLayout());

formContainer.setOpaque(false);

formContainer.add(formPanel, BorderLayout.CENTER);

contentPanel.add(formContainer, BorderLayout.CENTER);

// Botón de login

loginButton = createButton("Acceder", new Color(123, 118, 118));

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));

buttonPanel.setOpaque(false);

buttonPanel.setBorder(BorderFactory.createEmptyBorder(20, 0, 0, 0));

buttonPanel.add(loginButton);

contentPanel.add(buttonPanel, BorderLayout.SOUTH);

mainLayeredPane.add(contentPanel, JLayeredPane.MODAL\_LAYER);

getContentPane().add(mainLayeredPane);

// Acción del botón de login

loginButton.addActionListener(e -> {

String username = usernameField.getText().trim();

String password = new String(passwordField.getPassword()).trim();

if (username.isEmpty() || password.isEmpty()) {

JOptionPane.showMessageDialog(this,

"Por favor complete todos los campos",

"Campos vacíos",

JOptionPane.WARNING\_MESSAGE);

return;

}

if (usuarioDAO.autenticarUsuario(username, password)) {

this.dispose();

Usuario usuario = usuarioDAO.obtenerUsuario(username);

String rol = usuario.getRol();

SwingUtilities.invokeLater(() -> {

menuprincipal menu = new menuprincipal(usuario);

menu.setBienvenida(username, rol);

ventanas controlador = new ventanas(menu, usuario);

menu.setVisible(true);

});

} else {

JOptionPane.showMessageDialog(this,

"Usuario o contraseña incorrectos",

"Error de autenticación",

JOptionPane.ERROR\_MESSAGE);

}

});

}

// Método para mostrar el registro oculto

private void mostrarRegistroOculto() {

JDialog dialog = new JDialog(this, "Registro Temporal", true);

dialog.setSize(300, 200);

dialog.setLocationRelativeTo(this);

JPanel panel = new JPanel(new GridLayout(4, 2, 5, 5));

panel.setBorder(new EmptyBorder(10, 10, 10, 10));

JTextField txtUser = new JTextField();

JPasswordField txtPass = new JPasswordField();

JComboBox<String> comboRol = new JComboBox<>(new String[]{"Trab"});

panel.add(new JLabel("Usuario:"));

panel.add(txtUser);

panel.add(new JLabel("Contraseña:"));

panel.add(txtPass);

panel.add(new JLabel("Rol:"));

panel.add(comboRol);

JButton btnRegistrar = new JButton("Registrar");

JButton btnCancelar = new JButton("Cancelar");

btnRegistrar.addActionListener(e -> {

String user = txtUser.getText().trim();

String pass = new String(txtPass.getPassword()).trim();

if (user.isEmpty() || pass.isEmpty()) {

JOptionPane.showMessageDialog(dialog, "Complete todos los campos", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

if (usuarioDAO.registrarUsuario(user, pass, (String)comboRol.getSelectedItem())) {

JOptionPane.showMessageDialog(dialog, "Usuario temporal creado", "Éxito", JOptionPane.INFORMATION\_MESSAGE);

dialog.dispose();

} else {

JOptionPane.showMessageDialog(dialog, "Error al registrar", "Error", JOptionPane.ERROR\_MESSAGE);

}

});

btnCancelar.addActionListener(e -> dialog.dispose());

JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

btnPanel.add(btnCancelar);

btnPanel.add(btnRegistrar);

dialog.getContentPane().add(panel, BorderLayout.CENTER);

dialog.getContentPane().add(btnPanel, BorderLayout.SOUTH);

dialog.setVisible(true);

}

private void showMessage(String title, String message, int messageType) {

JOptionPane.showMessageDialog(this, message, title, messageType);

}

private JPanel createCirclePanel(Color color) {

return new JPanel() {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.setColor(color);

int diameter = Math.min(getWidth(), getHeight());

int x = (getWidth() - diameter) / 2;

int y = (getHeight() - diameter) / 2;

g2d.fillOval(x, y, diameter, diameter);

}

@Override

public Dimension getPreferredSize() {

return new Dimension(255, 255);

}

};

}

// Métodos públicos para el controlador

public String getUsername() {

return usernameField.getText().trim();

}

public String getPassword() {

return new String(passwordField.getPassword()).trim();

}

public void clearPasswordField() {

passwordField.setText("");

}

public void setLoginListener(ActionListener listener) {

loginButton.addActionListener(listener);

}

// Métodos de estilo

private void styleTextField(JComponent field) {

field.setFont(new Font("Arial", Font.PLAIN, 14));

field.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createMatteBorder(0, 0, 1, 0, Color.GRAY),

BorderFactory.createEmptyBorder(5, 5, 5, 5)

));

}

private JButton createButton(String text, Color bgColor) {

JButton button = new JButton(text);

button.setPreferredSize(new Dimension(255, 30));

button.setContentAreaFilled(false);

button.setOpaque(true);

button.setBackground(bgColor);

button.setForeground(Color.WHITE);

button.setFont(new Font("Arial", Font.BOLD, 14));

button.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

button.setCursor(new Cursor(Cursor.HAND\_CURSOR));

return button;

}

private void loginExitoso(Usuario usuario) {

// Después de login exitoso, abrir el 'menuprincipal' pasando el 'usuario'

new menuprincipal(usuario).setVisible(true);

this.dispose(); // Cierra la ventana de Login

}

}

Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

# Menú principal

package Vista;

import javax.swing.\*;

import Controlador.ventanas;

import Modelo.Usuario;

import java.awt.\*;

import java.awt.geom.\*;

import java.awt.event.\*;

public class menuprincipal extends JFrame {

private JButton btnGestionUsuarios;

private JButton btnGestionClientes;

private JButton btnRegistroVentas;

private JButton btnReportes;

private JButton btnInventario;

private JButton btnCerrarSesion;

private JButton btnProveedores;

private JLabel lblUsuario;

private Usuario usuario;

public menuprincipal(Usuario usuario) {

this.usuario = usuario;

initComponents();

new ventanas(this , usuario); // <- Se pasa el usuario al controlador

}

public Usuario getUsuario() {

return usuario;

}

private void initComponents() {

setTitle("El Habanerito - Menú Principal");

setSize(930, 704);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setResizable(false);

// Panel principal con diseño similar al login

JLayeredPane mainLayeredPane = new JLayeredPane();

mainLayeredPane.setPreferredSize(new Dimension(900, 650));

JPanel backgroundPanel = new JPanel() {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

GradientPaint gradient = new GradientPaint(0, 0, new Color(245, 245, 245),

0, getHeight(), new Color(230, 230, 230));

g2d.setPaint(gradient);

g2d.fillRect(0, 0, getWidth(), getHeight());

}

};

backgroundPanel.setBounds(0, 0, 900, 650);

mainLayeredPane.add(backgroundPanel, JLayeredPane.DEFAULT\_LAYER);

// Círculos decorativos

addCirculosDecorativos(mainLayeredPane);

// Panel de contenido principal

JPanel contentPanel = new JPanel(new BorderLayout());

contentPanel.setOpaque(false);

contentPanel.setBounds(0, 0, 900, 650);

// Cabecera con el nombre de la tienda y logo

JPanel headerPanel = new JPanel(new BorderLayout());

headerPanel.setOpaque(false);

headerPanel.setBorder(BorderFactory.createEmptyBorder(30, 30, 30, 0));

// Panel para el logo y título

JPanel titlePanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 0));

titlePanel.setOpaque(false);

try {

ImageIcon originalIcon = new ImageIcon("imagen\\logo.png");

Image originalImage = originalIcon.getImage();

int logoHeight = 200;

int logoWidth = (int) ((double) originalIcon.getIconWidth() / originalIcon.getIconHeight() \* logoHeight);

Image resizedImage = originalImage.getScaledInstance(logoWidth, logoHeight, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage));

titlePanel.add(logo);

} catch (Exception e) {

System.err.println("Error cargando el logo: " + e.getMessage());

}

JLabel lblTitulo = new JLabel("El Habanerito");

lblTitulo.setFont(new Font("Arial", Font.BOLD, 28));

lblTitulo.setForeground(new Color(70, 70, 70));

titlePanel.add(lblTitulo);

lblUsuario = new JLabel("", SwingConstants.RIGHT);

lblUsuario.setFont(new Font("Arial", Font.PLAIN, 14));

lblUsuario.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 30));

headerPanel.add(titlePanel, BorderLayout.CENTER);

headerPanel.add(lblUsuario, BorderLayout.EAST);

contentPanel.add(headerPanel, BorderLayout.NORTH);

// Panel central con botones

JPanel centerPanel = crearPanelCentral();

contentPanel.add(centerPanel, BorderLayout.CENTER);

// Panel inferior con botón de cerrar sesión

JPanel footerPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

footerPanel.setOpaque(false);

footerPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 20, 30));

btnCerrarSesion = new JButton("Cerrar Sesión");

btnCerrarSesion.setFont(new Font("Arial", Font.PLAIN, 14));

btnCerrarSesion.setBackground(new Color(123, 118, 118));

btnCerrarSesion.setForeground(Color.WHITE);

btnCerrarSesion.setFocusPainted(false);

btnCerrarSesion.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

btnCerrarSesion.addActionListener(e -> {

this.dispose();

new Login().setVisible(true);

});

footerPanel.add(btnCerrarSesion);

contentPanel.add(footerPanel, BorderLayout.SOUTH);

mainLayeredPane.add(contentPanel, JLayeredPane.MODAL\_LAYER);

getContentPane().add(mainLayeredPane);

}

private void addCirculosDecorativos(JLayeredPane layeredPane) {

int circleSize = 500;

JPanel circleTopLeft = createCirclePanel(new Color(235, 215, 107, 100));

circleTopLeft.setBounds(-circleSize/2, -circleSize/2, circleSize, circleSize);

JPanel circleTopRight = createCirclePanel(new Color(216, 237, 88, 100));

circleTopRight.setBounds(900-circleSize/2, -circleSize/2, circleSize, circleSize);

JPanel circleBottomLeft = createCirclePanel(new Color(235, 184, 35, 100));

circleBottomLeft.setBounds(-circleSize/2, 650-circleSize/2, circleSize, circleSize);

JPanel circleBottomRight = createCirclePanel(new Color(241, 81, 17, 100));

circleBottomRight.setBounds(900-circleSize/2, 650-circleSize/2, circleSize, circleSize);

layeredPane.add(circleTopLeft, JLayeredPane.PALETTE\_LAYER);

layeredPane.add(circleTopRight, JLayeredPane.PALETTE\_LAYER);

layeredPane.add(circleBottomLeft, JLayeredPane.PALETTE\_LAYER);

layeredPane.add(circleBottomRight, JLayeredPane.PALETTE\_LAYER);

}

private JPanel createCirclePanel(Color color) {

return new JPanel() {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.setColor(color);

int diameter = Math.min(getWidth(), getHeight());

int x = (getWidth() - diameter) / 2;

int y = (getHeight() - diameter) / 2;

g2d.fillOval(x, y, diameter, diameter);

}

};

}

private JPanel crearPanelCentral() {

JPanel panel = new JPanel(new GridLayout(2, 3, 30, 30));

panel.setOpaque(false);

panel.setBorder(BorderFactory.createEmptyBorder(20, 50, 20, 50));

// Botones del menú principal

btnGestionUsuarios = crearBotonMenu("Gestión de Usuarios", Color.LIGHT\_GRAY);

btnGestionClientes = crearBotonMenu("Clientes", Color.LIGHT\_GRAY);

btnRegistroVentas = crearBotonMenu("Ventas", Color.LIGHT\_GRAY);

btnInventario = crearBotonMenu("Inventario", Color.LIGHT\_GRAY);

btnReportes = crearBotonMenu("Reportes", Color.LIGHT\_GRAY);

btnProveedores = crearBotonMenu("Proveedores", Color.LIGHT\_GRAY);

panel.add(btnGestionUsuarios);

panel.add(btnGestionClientes);

panel.add(btnRegistroVentas);

panel.add(btnInventario);

panel.add(btnReportes);

panel.add(btnProveedores);

return panel;

}

private JButton crearBotonMenu(String texto, Color color) {

JButton boton = new JButton(texto) {

@Override

protected void paintComponent(Graphics g) {

if (!isOpaque()) {

Graphics2D g2d = (Graphics2D) g.create();

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.setColor(getBackground());

g2d.fillRoundRect(0, 0, getWidth(), getHeight(), 30, 30);

g2d.setColor(getForeground());

FontMetrics fm = g2d.getFontMetrics();

Rectangle2D r = fm.getStringBounds(getText(), g2d);

int x = (this.getWidth() - (int) r.getWidth()) / 2;

int y = (this.getHeight() - (int) r.getHeight()) / 2 + fm.getAscent();

g2d.drawString(getText(), x, y);

g2d.dispose();

} else {

super.paintComponent(g);

}

}

@Override

protected void paintBorder(Graphics g) {

Graphics2D g2d = (Graphics2D) g.create();

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.setColor(getBackground().darker());

g2d.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 30, 30);

g2d.dispose();

}

};

boton.setFont(new Font("Arial", Font.BOLD, 18));

boton.setBackground(color);

boton.setForeground(Color.BLACK);

boton.setFocusPainted(false);

boton.setContentAreaFilled(false);

boton.setOpaque(false);

boton.setBorder(BorderFactory.createEmptyBorder(15, 10, 15, 10));

boton.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

boton.setBackground(color.brighter());

}

@Override

public void mouseExited(MouseEvent e) {

boton.setBackground(color);

}

});

return boton;

}

private JButton crearBotonFooter(String texto) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.PLAIN, 14));

boton.setBackground(new Color(123, 118, 118));

boton.setForeground(Color.WHITE);

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

return boton;

}

// Métodos para integración con controlador

public void setGestionUsuariosListener(ActionListener listener) {

btnGestionUsuarios.addActionListener(listener);

}

public void setGestionClientesListener(ActionListener listener) {

btnGestionClientes.addActionListener(listener);

}

public void setRegistroVentasListener(ActionListener listener) {

btnRegistroVentas.addActionListener(listener);

}

public void setInventarioListener(ActionListener listener) {

btnInventario.addActionListener(listener);

}

public void setReportesListener(ActionListener listener) {

btnReportes.addActionListener(listener);

}

public void setProveedoresListener(ActionListener listener) {

btnProveedores.addActionListener(listener);

}

public void setCerrarSesionListener(ActionListener listener) {

btnCerrarSesion.addActionListener(listener);

}

public void mostrarMensaje(String mensaje, String titulo, int tipo) {

JOptionPane.showMessageDialog(this, mensaje, titulo, tipo);

}

public void ocultarOpcionGestionUsuarios() {

this.btnGestionUsuarios.setVisible(false);

}

public void setBienvenida(String nombreUsuario, String rol) {

setTitle("El Habanerito - Menú Principal (" + nombreUsuario + ")");

lblUsuario.setText("Bienvenido: " + nombreUsuario + " (" + rol + ")");

}

public void configurarVisibilidadBotones(boolean mostrarAdmin) {

btnGestionUsuarios.setVisible(mostrarAdmin);

btnGestionUsuarios.setEnabled(mostrarAdmin);

btnReportes.setEnabled(mostrarAdmin);

}

public boolean mostrarConfirmacion(String mensaje, String titulo) {

int respuesta = JOptionPane.showConfirmDialog(

this,

mensaje,

titulo,

JOptionPane.YES\_NO\_OPTION

);

return respuesta == JOptionPane.YES\_OPTION;

}

}

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

package Controlador;

import Vista.\*;

import java.awt.event.\*;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import Vista.producto;

import Modelo.Cliente;

import Modelo.ClienteDAO;

import Modelo.ClienteDAOImpl;

import Modelo.Usuario;

public class ventanas {

private Usuario usuario;

private menuprincipal menu;

private gestionUsuario gestionUsuario;

private clientes clientes;

private producto producto;

private inventario inventario;

private reportes reportes;

private proveedores proveedores;

private ClienteDAO clienteDAO;

private Cliente cliente;

public ventanas(menuprincipal menu, Usuario usuario) {

this.menu = menu;

this.usuario = usuario;

inicializarEventos();

}

private void inicializarEventos() {

menu.setGestionUsuariosListener(e -> abrirGestionUsuarios());

menu.setGestionClientesListener(e -> abrirGestionClientes(usuario));

menu.setRegistroVentasListener(e -> abrirRegistroVentas());

menu.setInventarioListener(e -> abrirInventario());

menu.setReportesListener(e -> abrirReportes());

menu.setProveedoresListener(e -> abrirProveedores());

menu.setCerrarSesionListener(e -> cerrarSesion());

}

private void abrirGestionUsuarios() {

menu.dispose(); // Cierra la ventana actual (menu)

new gestionUsuario(usuario).setVisible(true); // Reemplaza con tu clase real

}

public void abrirGestionClientes(Usuario usuario) {

try {

ClienteDAO clienteDAO = new ClienteDAOImpl();

clientes vistaClientes = new clientes(usuario, clienteDAO);

// Asegurar que la ventana se configure correctamente

vistaClientes.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

vistaClientes.setLocationRelativeTo(null);

vistaClientes.setVisible(true);

menu.dispose();

} catch (Exception e) {

JOptionPane.showMessageDialog(null,

"Error al abrir clientes: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

e.printStackTrace(); // Para depuración

}

}

private void abrirRegistroVentas() {

if (producto == null || !producto.isVisible()) {

if (producto != null) {

producto.dispose();

}

producto = new producto(usuario);

producto.setVisible(true);

menu.dispose();

}

}

private void abrirInventario() {

menu.dispose(); // Cierra la ventana actual (menu)

new inventario(usuario).setVisible(true); // Reemplaza con tu clase real

}

public void abrirReportes() {

ReportesControlador controlador = new ReportesControlador(reportes, usuario);

reportes ventana = new reportes(usuario, controlador); // pásaselo

ventana.setVisible(true);

}

private void abrirProveedores() {

menu.dispose(); // Cierra la ventana actual (menu)

new proveedores(usuario).setVisible(true); // Reemplaza con tu clase real

}

private void cerrarSesion() {

menu.dispose(); // Cierra la ventana actual (menu)

new Login().setVisible(true);

}

}

# Venta

package Modelo;

import java.sql.Timestamp;

import java.util.Date;

import javax.swing.ImageIcon;

public class Producto {

// Atributos

private String id;

private String nombre;

private int cantidad;

private String fecha;

private double precioUnitario; // Precio unitario del producto

private String rutaImagen; // Ruta de la imagen del producto, si se necesita almacenar una ruta.

private ImageIcon imagen; // Objeto ImageIcon para la imagen

private Usuario usuario;

public int total;

private String descripcion;

private String categoria;

private String proveedor;

private int cantidadDisponible;

private int stockMinimo;

private int stockMaximo;

private double precioCompra;

private double precioVenta;

private boolean tieneIVA;

private double descuento;

private Date fechaIngreso;

private String estado; // "Activo", "Descontinuado", "Dañado"

private String imagenPath;

private String unidadMedida;

// Constructor

public Producto(String nombre, int cantidad, String fecha, double precioUnitario, String rutaImagen) {

this.nombre = nombre;

this.cantidad = cantidad;

this.fecha = fecha;

this.precioUnitario = precioUnitario;

this.rutaImagen = rutaImagen;

this.imagen = new ImageIcon(rutaImagen); // Cargar la imagen desde la ruta

}

public Producto(String nombre, double precioUnitario, int cantidad) {

this.nombre = nombre;

this.precioUnitario = precioUnitario;

this.cantidad = cantidad;

}

public Producto(Usuario usuario) {

this.usuario = usuario;

initComponents(); // o tu método de inicialización

}

public Producto(String id, String nombre, String descripcion, String categoria,

String proveedor, int cantidadDisponible, int stockMinimo,

int stockMaximo, double precioCompra, double precioVenta,

boolean tieneIVA, double descuento, Date fechaIngreso,

String estado, String imagenPath, String unidadMedida) {

this.id = id;

this.nombre = nombre;

this.descripcion = descripcion;

this.categoria = categoria;

this.proveedor = proveedor;

this.cantidadDisponible = cantidadDisponible;

this.stockMinimo = stockMinimo;

this.stockMaximo = stockMaximo;

this.precioCompra = precioCompra;

this.precioVenta = precioVenta;

this.tieneIVA = tieneIVA;

this.descuento = descuento;

this.fechaIngreso = fechaIngreso;

this.estado = estado;

this.imagenPath = imagenPath;

this.unidadMedida = unidadMedida;

}

public Producto(String id, String nombre, double precioVenta, int cantidad) {

this.id = id;

this.nombre = nombre;

this.precioVenta = precioVenta; // Usamos precioVenta como campo principal

this.precioUnitario = precioVenta; // Mantenemos ambos precios sincronizados

this.cantidad = cantidad;

}

private void initComponents() {

// TODO Auto-generated method stub

}

// Getters y Setters

public String getId() { return id; }

public void setId(String id) { this.id = id; }

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public int getCantidad() {

return cantidad;

}

public void setCantidad(int cantidad) {

this.cantidad = cantidad;

}

public String getFecha() {

return fecha;

}

public void setFecha(String fecha) {

this.fecha = fecha;

}

public double getPrecioUnitario() {

return precioUnitario;

}

public void setPrecioUnitario(double precioUnitario) {

this.precioUnitario = precioUnitario;

}

public String getRutaImagen() {

return rutaImagen;

}

public void setRutaImagen(String rutaImagen) {

this.rutaImagen = rutaImagen;

this.imagen = new ImageIcon(rutaImagen); // Actualiza la imagen cuando cambia la ruta

}

public ImageIcon getImagen() {

return imagen;

}

public double setTotal(double d) {

return precioUnitario \* cantidad;

}

public String getUnidadMedida() { return unidadMedida; }

public void setUnidadMedida(String unidadMedida) { this.unidadMedida = unidadMedida; }

public String getDescripcion() { return descripcion; }

public void setDescripcion(String descripcion) { this.descripcion = descripcion; }

public String getCategoria() { return categoria; }

public void setCategoria(String categoria) { this.categoria = categoria; }

public String getProveedor() { return proveedor; }

public void setProveedor(String proveedor) { this.proveedor = proveedor; }

public int getCantidadDisponible() { return cantidadDisponible; }

public void setCantidadDisponible(int cantidadDisponible) { this.cantidadDisponible = cantidadDisponible; }

public int getStockMinimo() { return stockMinimo; }

public void setStockMinimo(int stockMinimo) { this.stockMinimo = stockMinimo; }

public int getStockMaximo() { return stockMaximo; }

public void setStockMaximo(int stockMaximo) { this.stockMaximo = stockMaximo; }

public double getPrecioCompra() { return precioCompra; }

public void setPrecioCompra(double precioCompra) { this.precioCompra = precioCompra; }

public double getPrecioVenta() { return precioVenta; }

public void setPrecioVenta(double precioVenta) { this.precioVenta = precioVenta; }

public boolean isTieneIVA() { return tieneIVA; }

public void setTieneIVA(boolean tieneIVA) { this.tieneIVA = tieneIVA; }

public double getDescuento() { return descuento; }

public void setDescuento(double descuento) { this.descuento = descuento; }

public Date getFechaIngreso() { return fechaIngreso; }

public void setFechaIngreso(Date fechaIngreso) { this.fechaIngreso = fechaIngreso; }

public String getEstado() { return estado; }

public void setEstado(String estado) { this.estado = estado; }

public String getImagenPath() { return imagenPath; }

public void setImagenPath(String imagenPath) { this.imagenPath = imagenPath; }

public boolean necesitaReposicion() {

return cantidadDisponible <= stockMinimo;

}

public boolean tieneExcesoStock() {

return cantidadDisponible > stockMaximo;

}

public double getPrecioConDescuento() {

return precioVenta \* (1 - descuento / 100);

}

public double getPrecioConIVA() {

return tieneIVA ? getPrecioConDescuento() \* 1.16 : getPrecioConDescuento();

}

// Método toString (opcional, para representación legible)

@Override

public String toString() {

return "Producto: " + nombre + ", Cantidad: " + cantidad + ", Precio Unitario: $" + precioUnitario + ", Fecha de vencimiento: " + (fecha.isEmpty() ? "No disponible" : fecha);

}

public void setVisible(boolean b) {

// TODO Auto-generated method stub

}

public Timestamp getVersion() {

// TODO Auto-generated method stub

return null;

}

}

package Controlador;

import Modelo.Producto;

import java.util.ArrayList;

import java.util.List;

public class ProductoControlador {

private List<Producto> productos;

// Constructor

public ProductoControlador() {

this.productos = new ArrayList<>();

}

// Método para agregar un producto

public void agregarProducto(String nombre, int cantidad, String fecha, double precioUnitario, String rutaImagen) {

Producto nuevoProducto = new Producto(nombre, cantidad, fecha, precioUnitario, rutaImagen);

productos.add(nuevoProducto);

}

// Método para eliminar un producto (por nombre)

public boolean eliminarProducto(String nombre) {

for (Producto producto : productos) {

if (producto.getNombre().equalsIgnoreCase(nombre)) {

productos.remove(producto);

return true;

}

}

return false;

}

// Método para modificar un producto (por nombre)

public boolean modificarProducto(String nombre, int nuevaCantidad, String nuevaFecha, double nuevoPrecioUnitario, String nuevaRutaImagen) {

for (Producto producto : productos) {

if (producto.getNombre().equalsIgnoreCase(nombre)) {

producto.setCantidad(nuevaCantidad);

producto.setFecha(nuevaFecha);

producto.setPrecioUnitario(nuevoPrecioUnitario);

producto.setRutaImagen(nuevaRutaImagen);

return true;

}

}

return false;

}

// Método para obtener todos los productos

public List<Producto> obtenerProductos() {

return productos;

}

// Método para obtener un producto por su nombre

public Producto obtenerProductoPorNombre(String nombre) {

for (Producto producto : productos) {

if (producto.getNombre().equalsIgnoreCase(nombre)) {

return producto;

}

}

return null; // Si no se encuentra el producto

}

public void mostrarProductos() {

// TODO Auto-generated method stub

}

public boolean actualizarProducto(Producto producto) {

// TODO Auto-generated method stub

return false;

}

public boolean agregarProducto(Producto producto) {

// TODO Auto-generated method stub

return false;

}

}

package Modelo;

import java.sql.Timestamp;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

public class Venta {

private int id;

private Date fecha; // Debe incluir fecha y hora

private double total;

private double descuento;

private String metodoPago;

private double montoRecibido;

private Cliente cliente;

private List<Producto> productos;

// Constructores

public Venta() {

this.fecha = new Date(); // Fecha y hora actual por defecto

this.productos = new ArrayList<>();

}

public Venta(int id, Date fecha, double total, String metodoPago, List<Producto> productos) {

this.id = id;

this.fecha = fecha;

this.total = total;

this.metodoPago = metodoPago;

this.productos = productos;

}

public Venta(String id, Timestamp fecha, double total, String metodoPago, List<Producto> productos) {

this(Integer.parseInt(id), fecha, total, metodoPago, productos);

this.fecha = fecha;

this.total = total;

this.metodoPago = metodoPago;

this.productos = productos != null ? productos : new ArrayList<>(); // Garantiza lista no nula

}

// Getters y Setters para todos los campos

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public Timestamp getFechaTimestamp() {

return new Timestamp(fecha.getTime());

}

public void setFechaTimestamp(Timestamp fecha) {

this.fecha = new Date(fecha.getTime());

}

public Date getFecha() {

return fecha;

}

public void setFecha(Date fecha) {

this.fecha = fecha;

}

public double getTotal() {

return total;

}

public void setTotal(double total) {

this.total = total;

}

public double getDescuento() {

return descuento;

}

public void setDescuento(double descuento) {

this.descuento = descuento;

}

public String getMetodoPago() {

return metodoPago;

}

public void setMetodoPago(String metodoPago) {

this.metodoPago = metodoPago;

}

public double getMontoRecibido() {

return montoRecibido;

}

public void setMontoRecibido(double montoRecibido) {

this.montoRecibido = montoRecibido;

}

public Cliente getCliente() {

return cliente;

}

public void setCliente(Cliente cliente) {

this.cliente = cliente;

}

public List<Producto> getProductos() {

return productos;

}

public void setProductos(List<Producto> productos) {

this.productos = productos;

}

public void agregarProducto(Producto producto) {

this.productos.add(producto);

}

}

package Modelo;

import java.util.ArrayList;

import java.util.List;

public class Carrito {

private List<Producto> carrito = new ArrayList<>();

private double totalVenta = 0.0;

public void agregarProducto(String id, String nombre, double precio, int cantidad) {

// Buscar si el producto ya está en el carrito

for (Producto item : carrito) {

if (item.getId().equals(id)) {

item.setCantidad(item.getCantidad() + cantidad);

item.setPrecioVenta(precio); // Actualizamos el precio si cambió

item.setPrecioUnitario(precio); // Mantener sincronizados

actualizarTotal();

return;

}

}

// Si no existe, crear nuevo producto con el constructor simplificado

Producto nuevo = new Producto(id, nombre, precio, cantidad);

carrito.add(nuevo);

actualizarTotal();

}

public void actualizarTotal() {

double nuevoTotal = 0.0;

for (Producto item : carrito) {

if (item != null && item.getPrecioUnitario() >= 0 && item.getCantidad() > 0) {

nuevoTotal += item.getPrecioUnitario() \* item.getCantidad();

}

}

this.totalVenta = nuevoTotal;

}

public void cancelarVenta() {

carrito.clear();

totalVenta = 0.0;

}

public double getTotalVenta() {

return totalVenta;

}

public void setTotalVenta(double total) {

this.totalVenta = total;

}

public List<Producto> getProductos() {

return new ArrayList<>(carrito); // Devolver una copia para evitar modificaciones externas

}

public List<Producto> getCarrito() {

return carrito;

}

public boolean eliminarProducto(String idProducto) {

for (Producto producto : carrito) {

if (producto.getId().equals(idProducto)) {

carrito.remove(producto);

actualizarTotal();

return true;

}

}

return false;

}

}

package Controlador;

import Modelo.Carrito;

import Modelo.Cliente;

import Modelo.Producto;

import Modelo.Reportes;

import Modelo.Usuario;

import Modelo.ClienteDAO;

import Modelo.ClienteDAOImpl;

import Modelo.InventarioDAO;

import javax.swing.\*;

import javax.swing.JOptionPane;

import Modelo.Venta;

import com.itextpdf.text.BaseColor;

import com.itextpdf.text.Chunk;

import com.itextpdf.text.Document;

import com.itextpdf.text.Element;

import com.itextpdf.text.Font;

import com.itextpdf.text.FontFactory;

import com.itextpdf.text.Image;

import com.itextpdf.text.Paragraph;

import com.itextpdf.text.Rectangle;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

import com.itextpdf.text.Phrase;

import ConexionBD.ConexionAccess;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.Iterator;

import java.util.List;

import java.awt.Component;

import java.awt.Container;

import java.awt.Desktop;

import java.awt.Window;

public class VentaContro {

private Carrito carrito;

private InventarioDAO inventarioDAO = new InventarioDAO();

private ClienteDAO clienteDAO = new ClienteDAOImpl();

private Usuario usuarioActual; // Usuario actual

private PdfPTable totalesTable;

public VentaContro(Usuario usuario) {

if (usuario == null) {

throw new IllegalArgumentException("El usuario no puede ser nulo");

}

this.carrito = new Carrito();

this.usuarioActual = usuario;

this.inventarioDAO = new InventarioDAO();

this.clienteDAO = new ClienteDAOImpl();

}

public void agregarProducto(String id, String nombre, double precio, int cantidad) {

carrito.agregarProducto(id, nombre, precio, cantidad);

}

public void cancelarVenta() {

carrito.cancelarVenta();

}

public double getTotalVenta() {

return carrito.getTotalVenta();

}

public List<Producto> getProductosEnCarrito() {

return carrito.getCarrito();

}

public boolean procesarPago(String metodoPago, String montoTexto, String telefono) throws Exception {

// Validaciones iniciales

if (metodoPago == null || metodoPago.trim().isEmpty()) {

throw new Exception("Seleccione un método de pago");

}

// Corregir posible typo en "tarjeta"

metodoPago = metodoPago.toUpperCase().replace("TARGETA", "TARJETA");

// Procesar monto

double montoRecibido;

double total = carrito.getTotalVenta();

double descuento = calcularDescuento();

double totalConDescuento = total - descuento;

double descuentoPorPuntos = 0;

try {

montoRecibido = Double.parseDouble(montoTexto);

} catch (NumberFormatException e) {

throw new Exception("Monto inválido. Ingrese una cantidad numérica válida");

}

// ============ PROCESAR CLIENTE Y PUNTOS ============

Cliente cliente = null;

if (!telefono.trim().isEmpty()) {

cliente = clienteDAO.buscarPorTelefono(telefono);

if (cliente != null) {

int maxPuntosUsables = (int) (totalConDescuento \* 5);

int puntosParaUsar = Math.min(maxPuntosUsables, cliente.getPuntos());

double maxDescuento = puntosParaUsar / 5.0;

int opcion = JOptionPane.showConfirmDialog(null,

"Cliente: " + cliente.getNombre() + "\n" +

"Puntos disponibles: " + cliente.getPuntos() + "(" +(cliente.getPuntos() / 5) +

"$ de descuento)\n" +

"¿Desea aplicar el descuento por puntos?", "Descuento por puntos", JOptionPane.YES\_NO\_OPTION);

if (opcion == JOptionPane.YES\_OPTION) {

descuentoPorPuntos = maxDescuento;

totalConDescuento -= descuentoPorPuntos;

cliente.setPuntos(cliente.getPuntos() - puntosParaUsar);

clienteDAO.actualizarCliente(cliente);

if(metodoPago.equals("EFECTIVO") && montoRecibido > totalConDescuento) {

montoRecibido = totalConDescuento;

}

}

}

int puntosGanados = (int) (totalConDescuento / 20);

cliente.setPuntos(cliente.getPuntos()+ puntosGanados);

clienteDAO.actualizarCliente(cliente);

}

// Validaciones por método de pago

if (metodoPago.equals("EFECTIVO")) {

if (montoRecibido < totalConDescuento) {

double faltante = totalConDescuento - montoRecibido;

throw new Exception(String.format("Monto insuficiente. Faltan $%,.2f", faltante));

}

} else if (metodoPago.equals("TARJETA")) {

double diferencia = Math.abs(montoRecibido - totalConDescuento);

if (diferencia > 0.50) { // Permitir hasta 50 centavos de diferencia

throw new Exception(String.format(

"Para pagos con tarjeta, el monto debe ser aproximadamente $%,.2f",

totalConDescuento

));

}

montoRecibido = totalConDescuento; // Usar el monto exacto

} else {

throw new Exception("Método de pago no válido");

}

// ============ VERIFICACIÓN DE STOCK ============

Connection connStock = null;

try {

connStock = ConexionAccess.conectar();

connStock.setAutoCommit(false);

// Verificar stock para todos los productos

String sqlVerificar = "SELECT p.id, p.nombre, p.cantidad\_disponible " +

"FROM Productos p " +

"WHERE p.id = ?";

try (PreparedStatement pstmt = connStock.prepareStatement(sqlVerificar)) {

for (Producto producto : carrito.getProductos()) {

pstmt.setString(1, producto.getId());

ResultSet rs = pstmt.executeQuery();

if (!rs.next()) {

throw new Exception("Producto no encontrado: " + producto.getNombre());

}

int stockDisponible = rs.getInt("cantidad\_disponible");

if (stockDisponible < producto.getCantidad()) {

throw new Exception("Stock insuficiente para " + rs.getString("nombre") +

". Disponible: " + stockDisponible +

", Solicitado: " + producto.getCantidad());

}

}

}

connStock.commit();

} catch (SQLException e) {

if (connStock != null) {

try { connStock.rollback(); } catch (SQLException ex) {}

}

throw new Exception("Error al verificar disponibilidad: " + e.getMessage());

} finally {

if (connStock != null) {

try { connStock.close(); } catch (SQLException e) {}

}

}

// ============ CREAR OBJETO VENTA CON TODOS LOS DATOS ============

Venta venta = new Venta();

venta.setFecha(new Date());

venta.setTotal(totalConDescuento);

venta.setMetodoPago(metodoPago);

venta.setMontoRecibido(montoRecibido); // Asegurar que tiene el monto correcto

venta.setDescuento(descuento + descuentoPorPuntos);

venta.setProductos(new ArrayList<>(carrito.getProductos()));

if (cliente != null) {

venta.setCliente(cliente);

} // Procesar puntos del cliente

// Registrar en BD

try {

registrarVentaEnBD(venta);

generarTicketVenta(venta);

carrito.cancelarVenta();

return true;

} catch (SQLException e) {

if (cliente != null && descuentoPorPuntos > 0) {

cliente.setPuntos(cliente.getPuntos()+ (int) (descuentoPorPuntos \* 5));

clienteDAO.actualizarCliente(cliente);

}

throw new Exception("Error al registrar la venta: " + e.getMessage());

}

}

// Método auxiliar para calcular descuentos

private double calcularDescuento() {

double descuento = 0;

// Ejemplo: 10% de descuento para compras mayores a $1000

double total = carrito.getTotalVenta();

if (total > 1000) {

descuento = total \* 0.10;

}

return descuento;

}

public void generarTicketVenta(Venta venta) {

try {

// Crear la carpeta tickets si no existe

File carpetaTickets = new File("tickets");

if (!carpetaTickets.exists()) {

boolean carpetaCreada = carpetaTickets.mkdirs();

if (!carpetaCreada) {

throw new IOException("No se pudo crear la carpeta 'tickets'");

}

}

// Configurar nombre de archivo

String filename = generarNombreArchivoTicket(venta);

// Configuración del documento

Document document = new Document(new Rectangle(226f, 800f), 5, 5, 5, 5);

PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(filename));

document.open();

// Fuentes personalizadas

Font fontTitulo = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 14, BaseColor.BLACK);

Font fontSubtitulo = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 10, BaseColor.BLACK);

Font fontNormal = FontFactory.getFont(FontFactory.HELVETICA, 8, BaseColor.BLACK);

Font fontNegrita = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 7, BaseColor.BLACK);

Font fontImporte = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 9, BaseColor.BLACK);

Font fontPequeno = FontFactory.getFont(FontFactory.HELVETICA, 7, BaseColor.BLACK);

// Logo del negocio (opcional)

try {

Image logo = Image.getInstance("imagen/logo.png");

logo.scaleToFit(100, 60);

logo.setAlignment(Element.ALIGN\_CENTER);

document.add(logo);

} catch (Exception e) {

System.out.println("Logo no encontrado, continuando sin él");

}

// Encabezado

Paragraph titulo = new Paragraph("EL HABANERITO", fontTitulo);

titulo.setAlignment(Element.ALIGN\_CENTER);

document.add(titulo);

Paragraph subtitulo = new Paragraph("Ticket de Venta #" + venta.getId(), fontSubtitulo);

subtitulo.setAlignment(Element.ALIGN\_CENTER);

document.add(subtitulo);

// Información de la venta

PdfPTable infoTable = new PdfPTable(2);

infoTable.setWidthPercentage(100);

infoTable.setWidths(new float[]{1, 2});

addCell(infoTable, "Fecha y Hora:", fontNegrita);

addCell(infoTable, new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(venta.getFecha()), fontNormal);

addCell(infoTable, "Atendió:", fontNegrita);

addCell(infoTable, usuarioActual.getUsername(), fontNormal);

// Información del cliente y puntos (si existe)

if (venta.getCliente() != null) {

addCell(infoTable, "Cliente:", fontNegrita);

addCell(infoTable, venta.getCliente().getNombre(), fontNormal);

addCell(infoTable, "Teléfono:", fontNegrita);

addCell(infoTable, venta.getCliente().getTelefono(), fontNormal);

// Calcular puntos antes de la compra (puntos actuales + puntos ganados en esta compra)

int puntosGanados = (int)(venta.getTotal() / 20);

int puntosAntes = venta.getCliente().getPuntos() - puntosGanados;

// Calcular puntos usados (si aplica)

double descuentoNormal = carrito.getTotalVenta() - (venta.getTotal() + venta.getDescuento());

double descuentoPuntos = venta.getDescuento() - descuentoNormal;

addCell(infoTable, "Puntos antes:", fontNegrita);

addCell(infoTable, String.valueOf(puntosAntes), fontNormal);

if (descuentoPuntos > 0) {

addCell(infoTable, "Puntos usados:", fontNegrita);

addCell(infoTable, String.valueOf((int)(descuentoPuntos \* 5)), fontNormal);

}

addCell(infoTable, "Puntos ganados:", fontNegrita);

addCell(infoTable, String.valueOf(puntosGanados), fontNormal);

addCell(infoTable, "Puntos actuales:", fontNegrita);

addCell(infoTable, String.valueOf(venta.getCliente().getPuntos()), fontNormal);

}

document.add(infoTable);

document.add(Chunk.NEWLINE);

// Tabla de productos

PdfPTable productosTable = new PdfPTable(5);

productosTable.setWidthPercentage(100);

productosTable.setWidths(new float[]{1, 3, 1, 1, 1});

// Encabezados de productos

addCell(productosTable, "CÓDIGO", fontNegrita);

addCell(productosTable, "NOMBRE", fontNegrita);

addCell(productosTable, "CANT.", fontNegrita);

addCell(productosTable, "P.UNIT.", fontNegrita);

addCell(productosTable, "IMPORTE", fontNegrita);

// Productos

double subtotal = 0;

for (Producto producto : venta.getProductos()) {

addCell(productosTable, producto.getId(), fontNormal);

addCell(productosTable, producto.getNombre(), fontNormal);

addCell(productosTable, String.valueOf(producto.getCantidad()), fontNormal);

addCell(productosTable, String.format("$%,.2f", producto.getPrecioUnitario()), fontNormal);

addCell(productosTable, String.format("$%,.2f", producto.getPrecioUnitario() \* producto.getCantidad()), fontNormal);

subtotal += producto.getPrecioUnitario() \* producto.getCantidad();

}

document.add(productosTable);

document.add(Chunk.NEWLINE);

// Totales

PdfPTable totalesTable = new PdfPTable(2);

totalesTable.setWidthPercentage(100);

totalesTable.setWidths(new float[]{1, 1});

addCell(totalesTable, "Subtotal:", fontNegrita);

addCell(totalesTable, String.format("$%,.2f", subtotal), fontImporte);

addCell(totalesTable, "IVA (16%):", fontNegrita);

addCell(totalesTable, String.format("$%,.2f", venta.getTotal() - subtotal), fontImporte);

// Mostrar descuento normal si existe

double descuentoNormal = carrito.getTotalVenta() - (venta.getTotal() + venta.getDescuento());

if (descuentoNormal > 0) {

addCell(totalesTable, "Descuento:", fontNegrita);

addCell(totalesTable, String.format("-$%,.2f", descuentoNormal), fontImporte);

}

// Mostrar descuento por puntos si existe

double descuentoPuntos = venta.getDescuento() - descuentoNormal;

if (descuentoPuntos > 0) {

addCell(totalesTable, "Descuento por puntos:", fontNegrita);

addCell(totalesTable, String.format("-$%,.2f", descuentoPuntos), fontImporte);

}

addCell(totalesTable, "Total:", fontNegrita);

addCell(totalesTable, String.format("$%,.2f", venta.getTotal()), fontImporte);

addCell(totalesTable, "Método de pago:", fontNegrita);

addCell(totalesTable, venta.getMetodoPago(), fontImporte);

// Sección de pago

if (venta.getMetodoPago().equalsIgnoreCase("EFECTIVO")) {

addCell(totalesTable, "Monto recibido:", fontNegrita);

addCell(totalesTable, String.format("$%,.2f", venta.getMontoRecibido()), fontImporte);

addCell(totalesTable, "Cambio:", fontNegrita);

double cambio = venta.getMontoRecibido() - venta.getTotal();

addCell(totalesTable, String.format("$%,.2f", cambio), fontImporte);

} else if (venta.getMetodoPago().equalsIgnoreCase("TARJETA")) {

addCell(totalesTable, "Forma de pago:", fontNegrita);

addCell(totalesTable, "Tarjeta - Pago exacto", fontImporte);

}

document.add(totalesTable);

document.add(Chunk.NEWLINE);

// Mensaje de agradecimiento

Paragraph gracias = new Paragraph("¡Gracias por su preferencia!", fontSubtitulo);

gracias.setAlignment(Element.ALIGN\_CENTER);

document.add(gracias);

// Datos del negocio

Paragraph datosNegocio = new Paragraph(

"Av. Principal 123, Centro\n" +

"Tel: 555-123-4567\n" +

"RFC: HAB123456ABC\n" +

"Horario: L-V 9:00 a 20:00",

fontPequeno);

datosNegocio.setAlignment(Element.ALIGN\_CENTER);

document.add(datosNegocio);

document.close();

// Abrir el PDF automáticamente

abrirArchivoPDF(filename);

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException("Error al generar ticket: " + e.getMessage());

}

}

private String generarNombreArchivoTicket(Venta venta) {

// Asegurar que la carpeta existe

File carpeta = new File("tickets");

if (!carpeta.exists()) {

carpeta.mkdirs();

}

// Formato consistente: tickets/TICKET\_[ID]\_[FECHA].pdf

SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd");

return "tickets/TICKET\_" + venta.getId() + "\_" + sdf.format(venta.getFecha()) + ".pdf";

}

// Método para reimprimir un ticket existente

public void reimprimirTicket(String idVenta) throws Exception {

// 1. Buscar el archivo PDF original

File archivoTicket = buscarArchivoTicket(idVenta);

if (archivoTicket != null && archivoTicket.exists()) {

System.out.println("[DEBUG] Encontrado archivo existente: " + archivoTicket.getAbsolutePath());

abrirArchivoPDF(archivoTicket.getAbsolutePath());

return;

}

// 2. Si no existe, buscar en la BD y regenerar

System.out.println("[DEBUG] No se encontró archivo, generando desde BD...");

Venta venta = obtenerVentaDesdeBD(idVenta);

if (venta == null) {

throw new Exception("No se encontró la venta con ID: " + idVenta);

}

// 3. Generar el ticket y guardarlo

generarTicketVenta(venta);

// 4. Verificar que se creó correctamente

File nuevoTicket = new File(generarNombreArchivoTicket(venta));

if (!nuevoTicket.exists()) {

throw new Exception("No se pudo generar el archivo del ticket");

}

abrirArchivoPDF(nuevoTicket.getAbsolutePath());

}

private File buscarArchivoTicket(String idVenta) {

// Buscar en la carpeta tickets con diferentes patrones por si cambió el formato

File carpetaTickets = new File("tickets");

if (!carpetaTickets.exists()) {

return null;

}

// Patrones de búsqueda alternativos

String[] patrones = {

"TICKET\_" + idVenta + "\_\*.pdf",

"Ticket\_" + idVenta + "\_\*.pdf",

idVenta + "\_\*.pdf",

"\*" + idVenta + "\*.pdf"

};

for (String patron : patrones) {

File[] archivos = carpetaTickets.listFiles((dir, name) -> name.matches(patron.replace("\*", ".\*")));

if (archivos != null && archivos.length > 0) {

return archivos[0]; // Devuelve el primer archivo que coincida

}

}

return null;

}

// Método auxiliar para abrir el PDF

private void abrirArchivoPDF(String filePath) throws Exception {

File file = new File(filePath);

if (!file.exists()) {

throw new FileNotFoundException("Archivo no encontrado: " + filePath);

}

if (Desktop.isDesktopSupported()) {

Desktop.getDesktop().open(file);

} else {

// Alternativa para sistemas sin soporte Desktop

String os = System.getProperty("os.name").toLowerCase();

Runtime rt = Runtime.getRuntime();

if (os.contains("win")) {

rt.exec("rundll32 url.dll,FileProtocolHandler " + filePath);

} else if (os.contains("mac")) {

rt.exec("open " + filePath);

} else if (os.contains("nix") || os.contains("nux")) {

rt.exec("xdg-open " + filePath);

} else {

throw new UnsupportedOperationException("No se puede abrir el PDF en este sistema");

}

}

}

// Método auxiliar para agregar celdas a las tablas

private void addCell(PdfPTable table, String text, Font font) {

PdfPCell cell = new PdfPCell(new Phrase(text, font));

cell.setBorder(Rectangle.NO\_BORDER);

cell.setPadding(3);

table.addCell(cell);

}

private Venta obtenerVentaDesdeBD(String idVenta) throws SQLException {

Connection conn = null;

PreparedStatement pstmtVenta = null;

PreparedStatement pstmtDetalles = null;

ResultSet rsVenta = null;

ResultSet rsDetalles = null;

try {

conn = ConexionAccess.conectar();

// 1. Consulta mejorada para obtener más datos de la venta

String sqlVenta = "SELECT v.\*, u.username as nombre\_usuario " +

"FROM Ventas v " +

"LEFT JOIN Usuarios u ON v.id\_usuario = u.username " +

"WHERE v.id = ?";

pstmtVenta = conn.prepareStatement(sqlVenta);

pstmtVenta.setString(1, idVenta);

rsVenta = pstmtVenta.executeQuery();

if (!rsVenta.next()) {

throw new SQLException("No se encontró la venta con ID: " + idVenta);

}

Venta venta = new Venta();

venta.setId(rsVenta.getInt("id"));

venta.setFecha(rsVenta.getTimestamp("fecha"));

venta.setTotal(rsVenta.getDouble("total"));

venta.setMetodoPago(rsVenta.getString("metodo\_pago"));

venta.setDescuento(rsVenta.getDouble("descuento"));

venta.setMontoRecibido(rsVenta.getDouble("monto\_recibido"));

// Establecer el usuario que realizó la venta

if (usuarioActual == null) {

usuarioActual = new Usuario(sqlVenta, sqlVenta, sqlVenta);

usuarioActual.setUsername(rsVenta.getString("nombre\_usuario"));

}

// 2. Obtener cliente si existe

String idCliente = rsVenta.getString("id\_cliente");

if (idCliente != null && !idCliente.trim().isEmpty()) {

Cliente cliente = clienteDAO.buscarPorId(idCliente);

venta.setCliente(cliente);

}

// 3. Consulta mejorada para obtener detalles

String sqlDetalles = "SELECT dv.\*, p.nombre, p.precio\_venta " +

"FROM DetalleVenta dv " +

"JOIN Productos p ON dv.id\_producto = p.id " +

"WHERE dv.id\_venta = ?";

pstmtDetalles = conn.prepareStatement(sqlDetalles);

pstmtDetalles.setString(1, idVenta);

rsDetalles = pstmtDetalles.executeQuery();

List<Producto> productos = new ArrayList<>();

while (rsDetalles.next()) {

Producto producto = new Producto(usuarioActual);

producto.setId(rsDetalles.getString("id\_producto"));

producto.setNombre(rsDetalles.getString("nombre"));

producto.setPrecioUnitario(rsDetalles.getDouble("precio\_venta"));

producto.setCantidad(rsDetalles.getInt("cantidad"));

productos.add(producto);

}

venta.setProductos(productos);

return venta;

} finally {

// Cerrar recursos

if (rsDetalles != null) rsDetalles.close();

if (rsVenta != null) rsVenta.close();

if (pstmtDetalles != null) pstmtDetalles.close();

if (pstmtVenta != null) pstmtVenta.close();

if (conn != null) conn.close();

}

}

private Venta obtenerVentaRecienRegistrada() {

Connection conn = null;

PreparedStatement pstmtVenta = null;

PreparedStatement pstmtDetalle = null;

ResultSet rsVenta = null;

ResultSet rsDetalle = null;

try {

conn = ConexionAccess.conectar();

// 1. Obtener la última venta registrada (la más reciente)

String sqlVenta = "SELECT TOP 1 id, fecha, total, metodo\_pago FROM Ventas ORDER BY id DESC";

pstmtVenta = conn.prepareStatement(sqlVenta);

rsVenta = pstmtVenta.executeQuery();

if (!rsVenta.next()) {

throw new SQLException("No se pudo obtener la venta recién registrada");

}

int idVenta = rsVenta.getInt("id");

Date fecha = rsVenta.getDate("fecha");

double total = rsVenta.getDouble("total");

String metodoPago = rsVenta.getString("metodo\_pago");

// 2. Obtener los productos de esta venta

String sqlDetalle = "SELECT p.id, p.nombre, p.precio\_venta, dv.cantidad " +

"FROM DetalleVenta dv " +

"JOIN Productos p ON dv.id\_producto = p.id " +

"WHERE dv.id\_venta = ?";

pstmtDetalle = conn.prepareStatement(sqlDetalle);

pstmtDetalle.setInt(1, idVenta);

rsDetalle = pstmtDetalle.executeQuery();

List<Producto> productos = new ArrayList<>();

while (rsDetalle.next()) {

Producto producto = new Producto(

rsDetalle.getString("id"),

rsDetalle.getString("nombre"),

"", // descripción

"", // categoría

"", // proveedor

rsDetalle.getInt("cantidad"), // cantidad vendida

0, // stock mínimo

0, // stock máximo

0.0,// precio compra

rsDetalle.getDouble("precio\_venta"),

false, // tiene IVA

0.0, // descuento

null, // fecha ingreso

"", // estado

"", // imagen path

"" // unidad medida

);

producto.setCantidad(rsDetalle.getInt("cantidad"));

productos.add(producto);

}

return new Venta(idVenta, fecha, total, metodoPago, productos);

} catch (SQLException e) {

e.printStackTrace();

throw new RuntimeException("Error al obtener la venta recién registrada: " + e.getMessage());

} finally {

// Cerrar todos los recursos

try {

if (rsDetalle != null) rsDetalle.close();

if (rsVenta != null) rsVenta.close();

if (pstmtDetalle != null) pstmtDetalle.close();

if (pstmtVenta != null) pstmtVenta.close();

if (conn != null) conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

private void registrarVentaEnBD(Venta venta) throws SQLException {

Connection conn = null;

PreparedStatement pstmtVenta = null;

PreparedStatement pstmtDetalle = null;

ResultSet generatedKeys = null;

try {

// 1. Establecer conexión

conn = ConexionAccess.conectar();

// Verificar modo de solo lectura

if (conn.isReadOnly()) {

throw new SQLException("La base de datos está en modo solo lectura");

}

// Iniciar transacción (Access soporta transacciones básicas)

conn.setAutoCommit(false);

// 2. Verificar stock disponible para todos los productos

verificarStockDisponible(conn, venta.getProductos());

// 3. Registrar la venta principal

String sqlVenta = "INSERT INTO Ventas (fecha, total, metodo\_pago, id\_usuario, descuento, monto\_recibido) " +

"VALUES (?, ?, ?, ?, ?, ?)";

pstmtVenta = conn.prepareStatement(sqlVenta, Statement.RETURN\_GENERATED\_KEYS);

pstmtVenta.setTimestamp(1, new java.sql.Timestamp(venta.getFecha().getTime()));

pstmtVenta.setDouble(2, venta.getTotal());

pstmtVenta.setString(3, venta.getMetodoPago());

pstmtVenta.setString(4, usuarioActual.getUsername());

pstmtVenta.setDouble(5, venta.getDescuento());

pstmtVenta.setDouble(6, venta.getMontoRecibido());

int affectedRows = pstmtVenta.executeUpdate();

if (affectedRows == 0) {

throw new SQLException("No se pudo registrar la venta principal");

}

// 4. Obtener el ID generado (Access usa un enfoque especial)

generatedKeys = pstmtVenta.getGeneratedKeys();

if (!generatedKeys.next()) {

throw new SQLException("No se pudo obtener el ID de la venta generada");

}

int idVenta = generatedKeys.getInt(1);

venta.setId(idVenta);

// 5. Registrar detalles de venta

String sqlDetalle = "INSERT INTO DetalleVenta (id\_venta, id\_producto, cantidad, precio\_unitario, subtotal) " +

"VALUES (?, ?, ?, ?, ?)";

pstmtDetalle = conn.prepareStatement(sqlDetalle);

for (Producto producto : venta.getProductos()) {

if (producto.getId() == null || producto.getId().trim().isEmpty()) {

throw new SQLException("ID de producto inválido: " + producto.getNombre());

}

pstmtDetalle.setInt(1, idVenta);

pstmtDetalle.setString(2, producto.getId());

pstmtDetalle.setInt(3, producto.getCantidad());

pstmtDetalle.setDouble(4, producto.getPrecioUnitario());

pstmtDetalle.setDouble(5, producto.getPrecioUnitario() \* producto.getCantidad());

pstmtDetalle.addBatch();

}

// Ejecutar todos los detalles en lote

int[] resultadosDetalle = pstmtDetalle.executeBatch();

for (int resultado : resultadosDetalle) {

if (resultado == PreparedStatement.EXECUTE\_FAILED) {

throw new SQLException("Error al registrar detalles de venta");

}

}

// 6. Actualizar inventario

actualizarInventario(conn, venta.getProductos());

// 7. Confirmar transacción si todo fue exitoso

conn.commit();

} catch (SQLException e) {

// Revertir transacción en caso de error

if (conn != null) {

try {

conn.rollback();

} catch (SQLException ex) {

throw new SQLException("Error al revertir transacción: " + ex.getMessage(), e);

}

}

throw new SQLException("Error al registrar la venta: " + e.getMessage(), e);

} finally {

// Cerrar recursos en orden inverso

if (generatedKeys != null) {

try { generatedKeys.close(); } catch (SQLException e) { /\* ignorar \*/ }

}

if (pstmtDetalle != null) {

try { pstmtDetalle.close(); } catch (SQLException e) { /\* ignorar \*/ }

}

if (pstmtVenta != null) {

try { pstmtVenta.close(); } catch (SQLException e) { /\* ignorar \*/ }

}

if (conn != null) {

try {

conn.setAutoCommit(true); // Restaurar autocommit

conn.close();

} catch (SQLException e) { /\* ignorar \*/ }

}

}

}

private void verificarStockDisponible(Connection conn, List<Producto> productos) throws SQLException {

// Consulta modificada para Access (eliminando WITH (ROWLOCK, UPDLOCK))

String sql = "SELECT id, nombre, cantidad\_disponible FROM Productos WHERE id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

for (Producto producto : productos) {

pstmt.setString(1, producto.getId());

ResultSet rs = pstmt.executeQuery();

if (!rs.next()) {

throw new SQLException("Producto no encontrado: " + producto.getId());

}

int stockDisponible = rs.getInt("cantidad\_disponible");

if (stockDisponible < producto.getCantidad()) {

throw new SQLException(String.format(

"Stock insuficiente para %s. Disponible: %d, Solicitado: %d",

rs.getString("nombre"),

stockDisponible,

producto.getCantidad()

));

}

}

}

}

private void actualizarInventario(Connection conn, List<Producto> productos) throws SQLException {

String sql = "UPDATE Productos SET cantidad\_disponible = cantidad\_disponible - ? WHERE id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

for (Producto producto : productos) {

pstmt.setInt(1, producto.getCantidad());

pstmt.setString(2, producto.getId());

pstmt.addBatch();

}

int[] resultados = pstmt.executeBatch();

for (int i = 0; i < resultados.length; i++) {

if (resultados[i] == PreparedStatement.EXECUTE\_FAILED) {

throw new SQLException("Error al actualizar inventario para producto: " + productos.get(i).getId());

}

}

}

}

private void crearTablaVentasSiNoExiste(Connection conn) throws SQLException {

if (!tablaExiste(conn, "Ventas")) {

try (Statement stmt = conn.createStatement()) {

// Crear tabla Ventas con estructura básica

stmt.execute("CREATE TABLE Ventas (" +

"ID AUTOINCREMENT PRIMARY KEY, " +

"fecha DATETIME NOT NULL, " +

"total CURRENCY NOT NULL, " +

"descuento CURRENCY NOT NULL, " +

"metodo\_pago TEXT(50) NOT NULL, " +

"id\_usuario TEXT(50) NOT NULL)");

// Crear tabla DetalleVenta relacionada

stmt.execute("CREATE TABLE DetalleVenta (" +

"ID AUTOINCREMENT PRIMARY KEY, " +

"id\_venta INTEGER NOT NULL, " +

"id\_producto TEXT(50) NOT NULL, " +

"cantidad INTEGER NOT NULL, " +

"precio\_unitario CURRENCY NOT NULL, " +

"subtotal CURRENCY NOT NULL)");

// Crear relación entre tablas

stmt.execute("ALTER TABLE DetalleVenta ADD CONSTRAINT FK\_DetalleVenta\_Ventas " +

"FOREIGN KEY (id\_venta) REFERENCES Ventas (ID)");

System.out.println("Tablas Ventas y DetalleVenta creadas exitosamente");

}

}

}

private boolean tablaExiste(Connection conn, String nombreTabla) throws SQLException {

DatabaseMetaData meta = conn.getMetaData();

try (ResultSet rs = meta.getTables(null, null, nombreTabla, new String[] {"TABLE"})) {

return rs.next();

}

}

private void procesarPuntosCliente(String telefono, double total) throws Exception {

if (!telefono.trim().isEmpty()) {

Cliente cliente = clienteDAO.buscarPorTelefono(telefono);

int puntosGanados = (int)(total / 20); // 1 punto por cada $20

if (cliente != null) {

cliente.setPuntos(cliente.getPuntos() + puntosGanados);

clienteDAO.actualizarCliente(cliente);

}

}

}

public String generarResumenVenta() {

StringBuilder resumen = new StringBuilder();

resumen.append("Productos:\n");

for (Producto p : carrito.getProductos()) {

resumen.append(String.format("- %s x%d: $%.2f\n",

p.getNombre(), p.getCantidad(), p.getPrecioUnitario()));

}

resumen.append(String.format("\nTotal: $%.2f", carrito.getTotalVenta()));

return resumen.toString();

}

public void setTotalVenta(double total) {

carrito.setTotalVenta(total);

}

public Carrito getCarrito() {

return carrito;

}

private void verificarStockDisponible() throws Exception {

try (Connection conn = ConexionAccess.conectar()) {

conn.setAutoCommit(false);

for (Producto producto : carrito.getProductos()) {

String sql = "SELECT cantidad\_disponible FROM Productos WHERE id = ? WITH (ROWLOCK, UPDLOCK)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, producto.getId());

ResultSet rs = pstmt.executeQuery();

if (!rs.next()) {

throw new Exception("Producto no encontrado: " + producto.getNombre());

}

int stockDisponible = rs.getInt("cantidad\_disponible");

if (stockDisponible < producto.getCantidad()) {

throw new Exception("Stock insuficiente para el producto " + producto.getNombre() +

". Disponible: " + stockDisponible +

", Solicitado: " + producto.getCantidad());

}

}

}

conn.commit();

} catch (SQLException e) {

throw new Exception("Error al verificar stock: " + e.getMessage());

}

}

private boolean verificarYReservarStock(List<Producto> productos) {

Connection conn = null;

try {

conn = ConexionAccess.conectar();

conn.setAutoCommit(false);

// Primero verificar

for (Producto p : productos) {

String sqlCheck = "SELECT cantidad\_disponible FROM Productos WHERE id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sqlCheck)) {

pstmt.setString(1, p.getId());

ResultSet rs = pstmt.executeQuery();

if (!rs.next() || rs.getInt(1) < p.getCantidad()) {

conn.rollback();

return false;

}

}

}

// Luego reservar (actualizar)

String sqlUpdate = "UPDATE Productos SET cantidad\_disponible = cantidad\_disponible - ? WHERE id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sqlUpdate)) {

for (Producto p : productos) {

pstmt.setInt(1, p.getCantidad());

pstmt.setString(2, p.getId());

pstmt.addBatch();

}

int[] results = pstmt.executeBatch();

for (int r : results) {

if (r <= 0) {

conn.rollback();

return false;

}

}

}

conn.commit();

return true;

} catch (SQLException e) {

if (conn != null) {

try { conn.rollback(); } catch (SQLException ex) {}

}

return false;

} finally {

if (conn != null) {

try { conn.close(); } catch (SQLException e) {}

}

}

}

public Producto eliminarProductoDelCarrito(String nombreProducto) throws SQLException {

// Buscar el producto en el carrito

Producto producto = getProductoEnCarrito(nombreProducto);

if (producto != null) {

// 1. Eliminar del carrito

carrito.getProductos().remove(producto);

// 2. Recalcular total

carrito.actualizarTotal();

// 3. Devolver stock al inventario

inventarioDAO.actualizarStock(producto.getId(), producto.getCantidad());

return producto;

}

return null;

}

public Producto getProductoEnCarrito(String nombre) {

if (nombre == null || nombre.trim().isEmpty()) {

return null;

}

return carrito.getProductos().stream()

.filter(p -> p.getNombre().equalsIgnoreCase(nombre.trim()))

.findFirst()

.orElse(null);

}

}

package Controlador;

import Modelo.Producto;

import java.util.ArrayList;

import java.util.List;

public class ProductoControlador {

private List<Producto> productos;

// Constructor

public ProductoControlador() {

this.productos = new ArrayList<>();

}

// Método para agregar un producto

public void agregarProducto(String nombre, int cantidad, String fecha, double precioUnitario, String rutaImagen) {

Producto nuevoProducto = new Producto(nombre, cantidad, fecha, precioUnitario, rutaImagen);

productos.add(nuevoProducto);

}

// Método para eliminar un producto (por nombre)

public boolean eliminarProducto(String nombre) {

for (Producto producto : productos) {

if (producto.getNombre().equalsIgnoreCase(nombre)) {

productos.remove(producto);

return true;

}

}

return false;

}

// Método para modificar un producto (por nombre)

public boolean modificarProducto(String nombre, int nuevaCantidad, String nuevaFecha, double nuevoPrecioUnitario, String nuevaRutaImagen) {

for (Producto producto : productos) {

if (producto.getNombre().equalsIgnoreCase(nombre)) {

producto.setCantidad(nuevaCantidad);

producto.setFecha(nuevaFecha);

producto.setPrecioUnitario(nuevoPrecioUnitario);

producto.setRutaImagen(nuevaRutaImagen);

return true;

}

}

return false;

}

// Método para obtener todos los productos

public List<Producto> obtenerProductos() {

return productos;

}

// Método para obtener un producto por su nombre

public Producto obtenerProductoPorNombre(String nombre) {

for (Producto producto : productos) {

if (producto.getNombre().equalsIgnoreCase(nombre)) {

return producto;

}

}

return null; // Si no se encuentra el producto

}

public void mostrarProductos() {

// TODO Auto-generated method stub

}

public boolean actualizarProducto(Producto producto) {

// TODO Auto-generated method stub

return false;

}

public boolean agregarProducto(Producto producto) {

// TODO Auto-generated method stub

return false;

}

}

Interfaz de usuario gráfica, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

# Reportes

package Modelo;

import java.util.Date;

import java.util.List;

public class Reportes {

private String usuarioActual;

private Date fechaGeneracion;

private String tipoReporte; // "VENTAS", "INVENTARIO", "TICKET"

private Object datosReporte; // Datos del reporte

public Reportes(String usuarioActual) {

this.usuarioActual = usuarioActual;

this.fechaGeneracion = new Date();

}

// Mantener tus métodos existentes

public String getUsuarioActual() {

return usuarioActual;

}

public void setUsuarioActual(String usuarioActual) {

this.usuarioActual = usuarioActual;

}

// Nuevos métodos para manejar reportes

public void setTipoReporte(String tipo) {

this.tipoReporte = tipo;

}

public String getTipoReporte() {

return tipoReporte;

}

public void setDatosReporte(Object datos) {

this.datosReporte = datos;

}

public Object getDatosReporte() {

return datosReporte;

}

public Date getFechaGeneracion() {

return fechaGeneracion;

}

}

package Controlador;

import Modelo.\*;

import Vista.reportes;

import com.itextpdf.text.\*;

import com.itextpdf.text.Font;

import com.itextpdf.text.Image;

import com.itextpdf.text.Rectangle;

import com.itextpdf.text.pdf.\*;

import org.apache.pdfbox.io.RandomAccessFile;

import org.apache.poi.ss.usermodel.\*;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.text.SimpleDateFormat;

import java.util.List;

import ConexionBD.ConexionAccess;

import Vista.ReporteClientePanel;

import Vista.ReporteInventarioPanel;

import Vista.ReporteProveedoresPanel;

import Vista.ReporteVentasPanel;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.JFreeChart;

import org.jfree.chart.plot.CategoryPlot;

import org.jfree.chart.plot.PlotOrientation;

import org.jfree.data.category.DefaultCategoryDataset;

import org.jfree.data.general.DefaultPieDataset;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.SerializationFeature;

import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;

import javax.swing.\*;

import java.awt.\*;

import java.awt.Desktop;

import java.awt.image.BufferedImage;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.\*;

import java.text.NumberFormat;

import java.util.\*;

import java.util.Date;

import java.sql.\*;

import java.util.List;

import java.util.stream.Collectors;

import org.apache.poi.ss.usermodel.\*;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import org.apache.poi.ss.usermodel.Workbook;

import org.apache.poi.ss.util.CellRangeAddress;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Cell;

import org.apache.poi.ss.usermodel.CellStyle;

import org.apache.poi.ss.usermodel.IndexedColors;

public class ReportesControlador {

private reportes vista;

private ReporteVentasPanel panelVentas;

private final Reportes modelo;

private String tipoReporteActual = "VENTAS"; // Por defecto

private Document document;

private PDFExporter pdfExporter;

private InventarioDAO inventarioDAO;

private ReporteInventarioPanel panelInventario;

private ClienteDAO clienteDAO;

private ReporteClientePanel panelClientes;

private ProveedorDAO proveedorDAO;

private ReporteProveedoresPanel panelProveedores;

private Usuario usuario;

public ReportesControlador(reportes vista, Usuario usuario) {

this.usuario = usuario;

this.vista = vista;

this.modelo = new Reportes(usuario.getUsername());

this.pdfExporter = new PDFExporter(this);

this.inventarioDAO = new InventarioDAO();

this.clienteDAO = new ClienteDAOImpl();

this.proveedorDAO = new ProveedorDAO();

}

public void setPanelVentas(ReporteVentasPanel panelVentas) {

this.panelVentas = panelVentas;

System.out.println("[DEBUG] Panel de ventas asignado al controlador");

}

public void setPanelInventario(ReporteInventarioPanel panelInventario) {

this.panelInventario = panelInventario;

System.out.println("[DEBUG] Panel de inventario asignado al controlador");

}

public void setPanelClientes(ReporteClientePanel panelClientes) {

this.panelClientes = panelClientes;

System.out.println("[DEBUG] Panel de clientes asignado al controlador");

}

public void setPanelProveedores(ReporteProveedoresPanel panelProveedores) {

this.panelProveedores = panelProveedores;

System.out.println("[DEBUG] Panel de proveedores asignado al controlador");

}

public void inicializarPaneles(Usuario usuario) {

System.out.println("[DEBUG] Inicializando paneles - Fase 2");

this.panelVentas = new ReporteVentasPanel(usuario, this);

}

private void verificarConexionBD() {

try (Connection testConn = ConexionAccess.conectar()) {

System.out.println("[DEBUG] Conexión a BD exitosa");

} catch (SQLException e) {

System.err.println("[ERROR] No se pudo conectar a la BD: " + e.getMessage());

panelVentas.mostrarError("Error de conexión a la base de datos");

}

}

public void filtrarReporte() {

try {

// 1. Obtener fechas del panel

Date fechaInicio = panelVentas.getFechaInicio();

Date fechaFin = panelVentas.getFechaFin();

// Si no hay fecha de inicio, usar fecha muy antigua

if (fechaInicio == null) {

Calendar cal = Calendar.getInstance();

cal.set(2000, 0, 1); // 1 de enero del 2000

fechaInicio = cal.getTime();

System.out.println("[DEBUG] Fecha inicio ajustada a valor por defecto: " + fechaInicio);

}

// 2. Ajustar fecha fin para incluir todo el día

if (fechaFin != null) {

Calendar cal = Calendar.getInstance();

cal.setTime(fechaFin);

cal.set(Calendar.HOUR\_OF\_DAY, 23);

cal.set(Calendar.MINUTE, 59);

cal.set(Calendar.SECOND, 59);

fechaFin = cal.getTime();

} else {

fechaFin = new Date(); // Fecha actual

}

System.out.println("[DEBUG] Fechas seleccionadas - Inicio: " + fechaInicio + " - Fin: " + fechaFin);

// 3. Obtener datos de la base de datos

List<Venta> ventas = obtenerVentasDesdeBD(fechaInicio, fechaFin);

System.out.println("[DEBUG] Ventas obtenidas: " + ventas.size());

// 4. Actualizar vista con los datos

actualizarVistaConDatos(ventas);

} catch (SQLException e) {

System.err.println("[ERROR] Al obtener ventas: " + e.getMessage());

panelVentas.mostrarError("Error al cargar datos: " + e.getMessage());

}

}

public void actualizarVistaConDatos(List<Venta> ventas) {

SwingUtilities.invokeLater(() -> {

try {

System.out.println("[DEBUG] Actualizando gráfico con " + ventas.size() + " ventas");

// Actualizar componentes principales

panelVentas.actualizarResumen(ventas);

panelVentas.mostrarDetalleVentas(ventas);

// Calcular y mostrar estadísticas

Map<String, Object> stats = generarEstadisticasVentas(ventas);

panelVentas.actualizarEstadisticas(

String.format("$%,.2f", stats.get("totalVentas")),

ventas.size(),

((Number)stats.get("totalProductos")).intValue(),

String.format("$%,.2f", (double)stats.get("totalVentas")/ventas.size())

);

// Actualizar gráficos secundarios

actualizarGraficosSecundarios(ventas, stats);

} catch (Exception e) {

System.err.println("[ERROR] Al actualizar vista: " + e.getMessage());

panelVentas.mostrarError("Error al mostrar datos: " + e.getMessage());

}

});

}

private boolean esMismoDia(Date fecha1, Date fecha2) {

Calendar cal1 = Calendar.getInstance();

Calendar cal2 = Calendar.getInstance();

cal1.setTime(fecha1);

cal2.setTime(fecha2);

return cal1.get(Calendar.YEAR) == cal2.get(Calendar.YEAR) &&

cal1.get(Calendar.MONTH) == cal2.get(Calendar.MONTH) &&

cal1.get(Calendar.DAY\_OF\_MONTH) == cal2.get(Calendar.DAY\_OF\_MONTH);

}

private void ajustarFechaFin(Date fechaFin) {

if (fechaFin != null) {

Calendar cal = Calendar.getInstance();

cal.setTime(fechaFin);

cal.set(Calendar.HOUR\_OF\_DAY, 23);

cal.set(Calendar.MINUTE, 59);

cal.set(Calendar.SECOND, 59);

fechaFin = cal.getTime();

}

}

private String formatCurrency(double value) {

return NumberFormat.getCurrencyInstance().format(value);

}

private void actualizarGraficosSecundarios(List<Venta> ventas, Map<String, Object> stats) {

// Actualizar gráfico de métodos de pago

if (panelVentas.getGraficoMetodosPago() != null) {

DefaultPieDataset datasetMetodos = new DefaultPieDataset();

((Map<String, Double>) stats.get("metodosPago")).forEach(datasetMetodos::setValue);

JFreeChart chartMetodos = ChartFactory.createPieChart(

"Métodos de Pago",

datasetMetodos,

true, true, false

);

panelVentas.getGraficoMetodosPago().setChart(chartMetodos);

}

// Actualizar gráfico de productos más vendidos

if (panelVentas.getGraficoProductos() != null) {

DefaultCategoryDataset datasetProductos = new DefaultCategoryDataset();

((Map<String, Long>) stats.get("productosVendidos")).entrySet().stream()

.sorted(Map.Entry.<String, Long>comparingByValue().reversed())

.limit(10)

.forEach(e -> datasetProductos.addValue(e.getValue(), "Ventas", e.getKey()));

JFreeChart chartProductos = ChartFactory.createBarChart(

"Top 10 Productos",

"Productos",

"Unidades Vendidas",

datasetProductos,

PlotOrientation.VERTICAL,

true, true, false

);

panelVentas.getGraficoProductos().setChart(chartProductos);

}

}

public List<Venta> obtenerVentasDesdeBD(Date fechaInicio, Date fechaFin) throws SQLException {

List<Venta> ventas = new ArrayList<>();

Connection conn = null;

PreparedStatement pstmtVentas = null;

PreparedStatement pstmtDetalles = null;

ResultSet rsVentas = null;

ResultSet rsDetalles = null;

try {

// 1. Obtener conexión

conn = ConexionAccess.conectar();

conn.setAutoCommit(false); // Usamos transacción

// 2. Consulta SQL para ventas principales

String sqlVentas = construirQueryVentas(fechaInicio, fechaFin);

pstmtVentas = conn.prepareStatement(sqlVentas);

// 3. Establecer parámetros de fecha

establecerParametrosFecha(pstmtVentas, fechaInicio, fechaFin);

// 4. Ejecutar consulta de ventas

rsVentas = pstmtVentas.executeQuery();

// 5. Procesar resultados

while (rsVentas.next()) {

Venta venta = mapearVentaDesdeResultSet(rsVentas);

// 6. Obtener detalles de la venta

String sqlDetalles = "SELECT p.id, p.nombre, p.precio\_venta, dv.cantidad " +

"FROM DetalleVenta dv " +

"JOIN Productos p ON dv.id\_producto = p.id " +

"WHERE dv.id\_venta = ?";

pstmtDetalles = conn.prepareStatement(sqlDetalles);

pstmtDetalles.setLong(1, venta.getId());

rsDetalles = pstmtDetalles.executeQuery();

// 7. Procesar detalles

while (rsDetalles.next()) {

Producto producto = new Producto(

rsDetalles.getString("id"),

rsDetalles.getString("nombre"),

"", // descripción

"", // categoría

"", // proveedor

rsDetalles.getInt("cantidad"),

0, // stock mínimo

0, // stock máximo

0.0, // precio compra

rsDetalles.getDouble("precio\_venta"),

false, // tiene\_iva

0.0, // porcentaje\_iva

null, // fecha\_vencimiento

"", // codigo\_barras

"", // unidad\_medida

"" // notas

);

venta.agregarProducto(producto);

}

ventas.add(venta);

cerrarResultSetYStatement(rsDetalles, pstmtDetalles);

}

conn.commit();

System.out.println("[BD] Ventas obtenidas: " + ventas.size());

return ventas;

} catch (SQLException e) {

if (conn != null) {

try {

conn.rollback();

} catch (SQLException ex) {

System.err.println("[BD] Error al hacer rollback: " + ex.getMessage());

}

}

System.err.println("[BD] Error al obtener ventas: " + e.getMessage());

throw e;

} finally {

cerrarResultSetYStatement(rsVentas, pstmtVentas);

cerrarResultSetYStatement(rsDetalles, pstmtDetalles);

if (conn != null) {

try {

conn.setAutoCommit(true); // Restaurar autocommit

conn.close();

} catch (SQLException e) {

System.err.println("[BD] Error al cerrar conexión: " + e.getMessage());

}

}

}

}

private String construirQueryVentas(Date fechaInicio, Date fechaFin) {

StringBuilder sql = new StringBuilder(

"SELECT id, fecha, total, metodo\_pago, descuento FROM Ventas WHERE 1=1"

);

if (fechaInicio != null) {

sql.append(" AND fecha >= ?");

}

if (fechaFin != null) {

sql.append(" AND fecha <= ?");

}

sql.append(" ORDER BY fecha DESC");

return sql.toString();

}

private void establecerParametrosFecha(PreparedStatement pstmt, Date fechaInicio, Date fechaFin)

throws SQLException {

int paramIndex = 1;

if (fechaInicio != null) {

pstmt.setTimestamp(paramIndex++, new Timestamp(fechaInicio.getTime()));

}

if (fechaFin != null) {

pstmt.setTimestamp(paramIndex, new Timestamp(fechaFin.getTime()));

}

}

private Venta mapearVentaDesdeResultSet(ResultSet rs) throws SQLException {

Venta venta = new Venta(

rs.getString("id"),

rs.getTimestamp("fecha"),

rs.getDouble("total"),

rs.getString("metodo\_pago"),

new ArrayList<>() // Productos se agregarán después

);

venta.setDescuento(rs.getDouble("descuento"));

return venta;

}

private void cerrarResultSetYStatement(ResultSet rs, Statement stmt) {

try {

if (rs != null) rs.close();

if (stmt != null) stmt.close();

} catch (SQLException e) {

System.err.println("[BD] Error al cerrar recursos: " + e.getMessage());

}

}

public void cargarProductosParaVenta(Venta venta) throws SQLException {

String sql = "SELECT p.id, p.nombre, p.precio\_venta, dv.cantidad " +

"FROM DetalleVenta dv " +

"JOIN Productos p ON dv.id\_producto = p.id " +

"WHERE dv.id\_venta = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setLong(1, venta.getId());

try (ResultSet rs = pstmt.executeQuery()) {

while (rs.next()) {

Producto producto = new Producto(

rs.getString("id"),

rs.getString("nombre"),

"", "", "",

rs.getInt("cantidad"),

0, 0, 0.0,

rs.getDouble("precio\_venta"),

false, 0.0, null, "", "", ""

);

venta.getProductos().add(producto);

}

}

}

}

public Map<String, Object> generarEstadisticasVentas(List<Venta> ventas) {

Map<String, Object> stats = new HashMap<>();

if (ventas == null || ventas.isEmpty()) {

return stats;

}

// Totales básicos

double totalVentas = ventas.stream().mapToDouble(Venta::getTotal).sum();

int totalTransacciones = ventas.size();

// Productos vendidos (con protección contra null)

long totalProductos = ventas.stream()

.filter(v -> v.getProductos() != null)

.flatMap(v -> v.getProductos().stream())

.count();

stats.put("totalVentas", totalVentas);

stats.put("totalTransacciones", totalTransacciones);

stats.put("totalProductos", totalProductos);

// Métodos de pago

Map<String, Double> metodosPago = ventas.stream()

.filter(v -> v.getMetodoPago() != null)

.collect(Collectors.groupingBy(

Venta::getMetodoPago,

Collectors.summingDouble(Venta::getTotal)

));

stats.put("metodosPago", metodosPago);

// Productos más vendidos

Map<String, Long> productosVendidos = ventas.stream()

.filter(v -> v.getProductos() != null)

.flatMap(v -> v.getProductos().stream())

.filter(p -> p.getNombre() != null)

.collect(Collectors.groupingBy(

Producto::getNombre,

Collectors.summingLong(Producto::getCantidad)

));

stats.put("productosVendidos", productosVendidos);

return stats;

}

public Map<String, Double> prepararDatosGrafica(List<Venta> ventas, String tipo) {

Map<String, Double> datos = new LinkedHashMap<>();

SimpleDateFormat sdf = obtenerFormatoFechaParaTipo(tipo);

if ("CATEGORIA".equals(tipo)) {

procesarDatosPorCategoria(ventas, datos);

} else {

procesarDatosPorFecha(ventas, datos, sdf);

}

return datos;

}

private SimpleDateFormat obtenerFormatoFechaParaTipo(String tipo) {

switch (tipo) {

case "DIARIO": return new SimpleDateFormat("dd/MM");

case "MENSUAL": return new SimpleDateFormat("MM/yyyy");

case "ANUAL": return new SimpleDateFormat("yyyy");

default: return new SimpleDateFormat("dd/MM/yyyy");

}

}

private void procesarDatosPorCategoria(List<Venta> ventas, Map<String, Double> datos) {

for (Venta venta : ventas) {

for (Producto producto : venta.getProductos()) {

String categoria = producto.getCategoria() != null ? producto.getCategoria() : "Sin categoría";

double valor = producto.getPrecioUnitario() \* producto.getCantidad();

datos.put(categoria, datos.getOrDefault(categoria, 0.0) + valor);

}

}

}

private void procesarDatosPorFecha(List<Venta> ventas, Map<String, Double> datos, SimpleDateFormat sdf) {

for (Venta venta : ventas) {

String clave = sdf.format(venta.getFecha());

datos.put(clave, datos.getOrDefault(clave, 0.0) + venta.getTotal());

}

}

public void exportarReporte() {

try {

// 1. Obtener parámetros básicos

String formato = panelVentas.getFormatoExportacion().toUpperCase();

String timestamp = new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date());

String nombreBase = "reporte\_" + tipoReporteActual.toLowerCase() + "\_" + timestamp;

// 2. Determinar la ubicación para guardar

File carpetaTipoReporte = crearEstructuraCarpetas();

// 3. Crear archivo con la ruta completa

String extension = obtenerExtension(formato);

File archivoReporte = new File(carpetaTipoReporte, nombreBase + extension);

// 4. Exportar según el formato

switch (formato) {

case "PDF":

exportarAPDF(archivoReporte.getAbsolutePath());

break;

case "EXCEL":

exportarAExcel(archivoReporte);

break;

case "HTML":

exportarAHTML(archivoReporte);

break;

case "CSV":

exportarACSV(archivoReporte);

break;

case "JSON":

exportarAJSON(archivoReporte);

break;

case "XML":

exportarAXML(archivoReporte);

break;

default:

mostrarErrorEnPanel("Formato no soportado: " + formato);

return;

}

// 5. Mostrar resultado y abrir archivo

manejarPostExportacion(archivoReporte, formato);

} catch (Exception e) {

mostrarErrorExportacion(e);

}

}

private File crearEstructuraCarpetas() throws IOException {

// Obtener la ruta del proyecto actual

String projectPath = System.getProperty("\"C:\\Users\\Anahi\\eclipse-workspace\\punto\_venta\_2\"");

File carpetaReportes = new File(projectPath, "Reportes");

if (!carpetaReportes.exists() && !carpetaReportes.mkdir()) {

throw new IOException("No se pudo crear la carpeta Reportes en: " + projectPath);

}

File carpetaTipoReporte = new File(carpetaReportes, tipoReporteActual);

if (!carpetaTipoReporte.exists() && !carpetaTipoReporte.mkdir()) {

throw new IOException("No se pudo crear la carpeta para " + tipoReporteActual);

}

return carpetaTipoReporte;

}

private String obtenerExtension(String formato) {

switch (formato.toUpperCase()) {

case "PDF": return ".pdf";

case "EXCEL": return ".xlsx";

case "HTML": return ".html";

case "CSV": return ".csv";

case "JSON": return ".json";

case "XML": return ".xml";

default: return ".txt";

}

}

private void exportarAJSON(File archivo) throws Exception {

List<Venta> ventas = obtenerVentasDesdeBD(panelVentas.getFechaInicio(), panelVentas.getFechaFin());

Map<String, Object> datosReporte = generarDatosReporte(ventas);

ObjectMapper mapper = new ObjectMapper();

mapper.enable(SerializationFeature.INDENT\_OUTPUT);

try (FileWriter writer = new FileWriter(archivo)) {

mapper.writeValue(writer, datosReporte);

}

}

private Map<String, Object> generarDatosReporte(List<Venta> ventas) {

Map<String, Object> reporte = new LinkedHashMap<>();

// Metadatos

reporte.put("tipo\_reporte", tipoReporteActual);

reporte.put("fecha\_generacion", new Date());

reporte.put("rango\_fechas", Map.of(

"inicio", panelVentas.getFechaInicio(),

"fin", panelVentas.getFechaFin()

));

// Datos de ventas

List<Map<String, Object>> ventasData = new ArrayList<>();

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");

for (Venta venta : ventas) {

Map<String, Object> ventaMap = new LinkedHashMap<>();

ventaMap.put("id", venta.getId());

ventaMap.put("fecha", sdf.format(venta.getFecha()));

ventaMap.put("total", venta.getTotal());

ventaMap.put("metodo\_pago", venta.getMetodoPago());

List<Map<String, Object>> productos = new ArrayList<>();

for (Producto producto : venta.getProductos()) {

productos.add(Map.of(

"nombre", producto.getNombre(),

"cantidad", producto.getCantidad(),

"precio\_unitario", producto.getPrecioUnitario(),

"total", producto.getPrecioUnitario() \* producto.getCantidad()

));

}

ventaMap.put("productos", productos);

ventasData.add(ventaMap);

}

reporte.put("ventas", ventasData);

// Estadísticas

Map<String, Object> stats = generarEstadisticasVentas(ventas);

reporte.put("estadisticas", stats);

return reporte;

}

private void exportarAXML(File archivo) throws Exception {

List<Venta> ventas = obtenerVentasDesdeBD(panelVentas.getFechaInicio(), panelVentas.getFechaFin());

try (PrintWriter writer = new PrintWriter(new FileOutputStream(archivo))) {

writer.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");

writer.println("<reporte tipo=\"" + tipoReporteActual + "\">");

writer.println(" <fechaGeneracion>" + new Date() + "</fechaGeneracion>");

writer.println(" <rango>");

writer.println(" <inicio>" + panelVentas.getFechaInicio() + "</inicio>");

writer.println(" <fin>" + panelVentas.getFechaFin() + "</fin>");

writer.println(" </rango>");

if (!ventas.isEmpty()) {

Map<String, Object> stats = generarEstadisticasVentas(ventas);

writer.println(" <resumen>");

writer.println(" <totalVentas>" + stats.get("totalVentas") + "</totalVentas>");

writer.println(" <cantidadVentas>" + ventas.size() + "</cantidadVentas>");

writer.println(" <productosVendidos>" + stats.get("totalProductos") + "</productosVendidos>");

writer.println(" <ticketPromedio>" + ((double)stats.get("totalVentas"))/ventas.size() + "</ticketPromedio>");

writer.println(" </resumen>");

writer.println(" <ventas>");

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");

for (Venta venta : ventas) {

writer.println(" <venta id=\"" + venta.getId() + "\" fecha=\"" + sdf.format(venta.getFecha()) +

"\" total=\"" + venta.getTotal() + "\">");

for (Producto producto : venta.getProductos()) {

writer.println(" <producto nombre=\"" + escapeXML(producto.getNombre()) + "\"" +

" cantidad=\"" + producto.getCantidad() + "\"" +

" precioUnitario=\"" + producto.getPrecioUnitario() + "\"" +

" total=\"" + (producto.getPrecioUnitario() \* producto.getCantidad()) + "\"/>");

}

writer.println(" </venta>");

}

writer.println(" </ventas>");

} else {

writer.println(" <mensaje>No hay datos para el período seleccionado</mensaje>");

}

writer.println("</reporte>");

}

}

private String escapeXML(String input) {

return input.replace("&", "&amp;")

.replace("<", "&lt;")

.replace(">", "&gt;")

.replace("\"", "&quot;")

.replace("'", "&apos;");

}

private void exportarACSV(File archivo) throws Exception {

List<Venta> ventas = obtenerVentasDesdeBD(panelVentas.getFechaInicio(), panelVentas.getFechaFin());

try (PrintWriter writer = new PrintWriter(new FileOutputStream(archivo))) {

// Encabezados

writer.println("ID\_Venta,Fecha,Producto,Cantidad,Precio\_Unitario,Total,Metodo\_Pago");

// Datos

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

for (Venta venta : ventas) {

for (Producto producto : venta.getProductos()) {

writer.println(String.format("%d,%s,\"%s\",%d,%.2f,%.2f,%s",

venta.getId(),

sdf.format(venta.getFecha()),

producto.getNombre().replace("\"", "\"\""),

producto.getCantidad(),

producto.getPrecioUnitario(),

producto.getPrecioUnitario() \* producto.getCantidad(),

venta.getMetodoPago()

));

}

}

}

}

public void exportarAExcel(File archivo) throws Exception {

List<Venta> ventas = obtenerVentasDesdeBD(panelVentas.getFechaInicio(), panelVentas.getFechaFin());

try (Workbook workbook = new XSSFWorkbook()) {

Sheet sheet = workbook.createSheet("Reporte de Ventas");

// Estilos

CellStyle headerStyle = crearEstiloEncabezado(workbook);

CellStyle titleStyle = crearEstiloTitulo(workbook);

CellStyle currencyStyle = crearEstiloMoneda(workbook);

CellStyle dateStyle = crearEstiloFecha(workbook);

int currentRow = 0;

// Logo e información

Row logoRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, logoRow, 0, 6, "EL HABANERITO - REPORTE DE VENTAS", titleStyle);

Row infoRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, infoRow, 0, 6, "Atendido por: " + modelo.getUsuarioActual(), null);

Row fechaRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, fechaRow, 0, 6, "Fecha: " + new SimpleDateFormat("yyyy-MM-dd HH:mm").format(new Date()), null);

currentRow++;

// Encabezados

Row headerRow = sheet.createRow(currentRow++);

String[] headers = {"ID Venta", "Fecha", "Producto", "Cantidad", "Precio Unitario", "Total", "Método Pago"};

for (int i = 0; i < headers.length; i++) {

Cell cell = headerRow.createCell(i);

cell.setCellValue(headers[i]);

cell.setCellStyle(headerStyle);

}

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");

// Mapa para totales por método de pago

Map<String, Double> totalesPorMetodo = new HashMap<>();

// Datos de ventas

for (Venta venta : ventas) {

if (venta.getProductos() == null || venta.getProductos().isEmpty()) continue;

for (Producto producto : venta.getProductos()) {

if (producto == null) continue;

Row row = sheet.createRow(currentRow++);

crearCelda(row, 0, venta.getId(), null);

crearCelda(row, 1, sdf.format(venta.getFecha()), dateStyle);

String nombreProducto = producto.getNombre() != null ? producto.getNombre() : "Sin nombre";

crearCelda(row, 2, nombreProducto, null);

crearCelda(row, 3, producto.getCantidad(), null);

crearCelda(row, 4, producto.getPrecioUnitario(), currencyStyle);

double total = producto.getPrecioUnitario() \* producto.getCantidad();

crearCelda(row, 5, total, currencyStyle);

String metodoPago = venta.getMetodoPago() != null ? venta.getMetodoPago() : "No especificado";

crearCelda(row, 6, metodoPago, null);

// Acumulamos total por método de pago

totalesPorMetodo.merge(metodoPago, total, Double::sum);

}

}

// Autoajustar columnas

for (int i = 0; i < headers.length; i++) {

sheet.autoSizeColumn(i);

sheet.setColumnWidth(i, Math.max(sheet.getColumnWidth(i), 3000));

}

// Espacio extra antes del resumen

currentRow++;

currentRow++;

// Título del resumen

Row resumenTitulo = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, resumenTitulo, 0, 6, "Resumen por Método de Pago", titleStyle);

// Encabezados de resumen

Row resumenHeader = sheet.createRow(currentRow++);

crearCelda(resumenHeader, 0, "Método de Pago", headerStyle);

crearCelda(resumenHeader, 1, "Total Ventas", headerStyle);

// Filas del resumen

for (Map.Entry<String, Double> entry : totalesPorMetodo.entrySet()) {

Row rowResumen = sheet.createRow(currentRow++);

crearCelda(rowResumen, 0, entry.getKey(), null);

crearCelda(rowResumen, 1, entry.getValue(), currencyStyle);

}

// Guardar archivo

try (FileOutputStream outputStream = new FileOutputStream(archivo)) {

workbook.write(outputStream);

}

}

}

// Métodos auxiliares para estilos y creación de celdas

private CellStyle crearEstiloEncabezado(Workbook workbook) {

CellStyle style = workbook.createCellStyle();

org.apache.poi.ss.usermodel.Font font = workbook.createFont();

font.setBold(true);

font.setColor(IndexedColors.WHITE.getIndex());

style.setFont(font);

style.setFillForegroundColor(IndexedColors.DARK\_BLUE.getIndex());

style.setFillPattern(FillPatternType.SOLID\_FOREGROUND);

style.setAlignment(HorizontalAlignment.CENTER);

style.setBorderBottom(BorderStyle.THIN);

style.setBorderTop(BorderStyle.THIN);

style.setBorderLeft(BorderStyle.THIN);

style.setBorderRight(BorderStyle.THIN);

return style;

}

private CellStyle crearEstiloTitulo(Workbook workbook) {

CellStyle style = workbook.createCellStyle();

org.apache.poi.ss.usermodel.Font font = workbook.createFont();

font.setBold(true);

font.setFontHeightInPoints((short)14);

style.setFont(font);

style.setAlignment(HorizontalAlignment.CENTER);

return style;

}

private CellStyle crearEstiloMoneda(Workbook workbook) {

CellStyle style = workbook.createCellStyle();

style.setDataFormat(workbook.createDataFormat().getFormat("$#,##0.00"));

return style;

}

private CellStyle crearEstiloFecha(Workbook workbook) {

CellStyle style = workbook.createCellStyle();

style.setDataFormat(workbook.createDataFormat().getFormat("dd/mm/yyyy hh:mm"));

return style;

}

private void crearCeldaConMergedRegion(Sheet sheet, Row row, int colInicio, int colFin, String valor, CellStyle estilo) {

Cell cell = row.createCell(colInicio);

cell.setCellValue(valor);

if (estilo != null) {

cell.setCellStyle(estilo);

}

sheet.addMergedRegion(new CellRangeAddress(row.getRowNum(), row.getRowNum(), colInicio, colFin));

}

private void crearCelda(Row row, int columna, Object valor, CellStyle estilo) {

Cell cell = row.createCell(columna);

if (valor == null) {

cell.setCellValue("");

} else if (valor instanceof Number) {

cell.setCellValue(((Number) valor).doubleValue());

} else if (valor instanceof Date) {

cell.setCellValue((Date) valor);

} else {

cell.setCellValue(valor.toString());

}

if (estilo != null) {

cell.setCellStyle(estilo);

}

}

private void exportarAHTML(File archivo) throws Exception {

List<Venta> ventas = obtenerVentasDesdeBD(panelVentas.getFechaInicio(), panelVentas.getFechaFin());

try (PrintWriter writer = new PrintWriter(new FileOutputStream(archivo))) {

writer.println("<!DOCTYPE html>");

writer.println("<html lang='es'>");

writer.println("<head>");

writer.println("<meta charset='UTF-8'>");

writer.println("<title>Reporte de " + tipoReporteActual + "</title>");

writer.println("<style>");

writer.println("body { font-family: Arial, sans-serif; margin: 20px; }");

writer.println("h1 { color: #2c3e50; }");

writer.println("table { width: 100%; border-collapse: collapse; margin-bottom: 20px; }");

writer.println("th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }");

writer.println("th { background-color: #f2f2f2; }");

writer.println(".resumen { background-color: #f8f9fa; padding: 15px; margin-bottom: 20px; }");

writer.println("</style>");

writer.println("</head>");

writer.println("<body>");

writer.println("<h1>Reporte de " + tipoReporteActual + "</h1>");

writer.println("<div class='resumen'>");

writer.println("<p><strong>Generado:</strong> " + new Date() + "</p>");

writer.println("<p><strong>Rango:</strong> " + panelVentas.getFechaInicio() + " - " + panelVentas.getFechaFin() + "</p>");

if (!ventas.isEmpty()) {

Map<String, Object> stats = generarEstadisticasVentas(ventas);

writer.println("<p><strong>Total Ventas:</strong> $" + String.format("%,.2f", stats.get("totalVentas")) + "</p>");

writer.println("<p><strong>Cantidad de Ventas:</strong> " + ventas.size() + "</p>");

writer.println("<p><strong>Productos Vendidos:</strong> " + stats.get("totalProductos") + "</p>");

writer.println("<p><strong>Ticket Promedio:</strong> $" +

String.format("%,.2f", ((double)stats.get("totalVentas"))/ventas.size()) + "</p>");

}

writer.println("</div>");

if (!ventas.isEmpty()) {

writer.println("<h2>Detalle de Ventas</h2>");

writer.println("<table>");

writer.println("<tr><th>ID Venta</th><th>Fecha</th><th>Producto</th><th>Cantidad</th><th>Precio Unitario</th><th>Total</th><th>Método Pago</th></tr>");

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");

for (Venta venta : ventas) {

for (Producto producto : venta.getProductos()) {

writer.println(String.format(

"<tr><td>%d</td><td>%s</td><td>%s</td><td>%d</td><td>$%.2f</td><td>$%.2f</td><td>%s</td></tr>",

venta.getId(),

sdf.format(venta.getFecha()),

producto.getNombre(),

producto.getCantidad(),

producto.getPrecioUnitario(),

producto.getPrecioUnitario() \* producto.getCantidad(),

venta.getMetodoPago()

));

}

}

writer.println("</table>");

} else {

writer.println("<p>No hay datos para el período seleccionado</p>");

}

writer.println("</body>");

writer.println("</html>");

}

}

private void manejarPostExportacion(File archivo, String formato) throws IOException {

// Verificar que el archivo se creó

if (!archivo.exists() || archivo.length() == 0) {

throw new IOException("El archivo no se generó correctamente");

}

// Mostrar mensaje de éxito

String mensajeExito = String.format(

"<html><b>Reporte exportado exitosamente</b><br>" +

"<b>Tipo:</b> %s<br>" +

"<b>Formato:</b> %s<br>" +

"<b>Ubicación:</b> %s</html>",

tipoReporteActual,

formato,

archivo.getAbsolutePath()

);

panelVentas.mostrarMensaje(mensajeExito);

// Intentar abrir el archivo o carpeta

abrirArchivoExportado(archivo, formato);

}

private void abrirArchivoExportado(File archivo, String formato) {

if (!Desktop.isDesktopSupported()) {

return;

}

try {

Desktop desktop = Desktop.getDesktop();

// Para PDF y HTML, abrir el archivo directamente

if (formato.equalsIgnoreCase("PDF") || formato.equalsIgnoreCase("HTML")) {

desktop.open(archivo);

}

// Para otros formatos, abrir la carpeta contenedora

else {

desktop.open(archivo.getParentFile());

}

} catch (Exception e) {

System.err.println("Error al abrir archivo: " + e.getMessage());

}

}

private void mostrarErrorExportacion(Exception e) {

String mensajeError = String.format(

"<html><b>Error al exportar reporte</b><br>" +

"<b>Tipo:</b> %s<br>" +

"<b>Error:</b> %s</html>",

tipoReporteActual,

e.getMessage()

);

panelVentas.mostrarError(mensajeError);

e.printStackTrace();

}

private void mostrarMensajeEnPanel(String mensaje) {

if (panelVentas != null) {

panelVentas.mostrarMensaje(mensaje);

} else if (vista != null) {

vista.mostrarMensaje(mensaje);

} else {

System.out.println(mensaje);

}

}

private void mostrarErrorEnPanel(String mensaje) {

if (panelVentas != null) {

panelVentas.mostrarError(mensaje);

} else if (vista != null) {

vista.mostrarError(mensaje);

} else {

System.err.println(mensaje);

}

}

private void exportarAXML(String filename) throws Exception {

try (PrintWriter writer = new PrintWriter(new FileOutputStream(filename))) {

writer.println("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");

writer.println("<reporte tipo=\"" + tipoReporteActual + "\">");

writer.println(" <fechaGeneracion>" + new Date() + "</fechaGeneracion>");

writer.println(" <rango>");

writer.println(" <inicio>" + panelVentas.getFechaInicio() + "</inicio>");

writer.println(" <fin>" + panelVentas.getFechaFin() + "</fin>");

writer.println(" </rango>");

if ("VENTAS".equals(tipoReporteActual)) {

List<Venta> ventas = obtenerVentasDesdeBD(panelVentas.getFechaInicio(), panelVentas.getFechaFin());

Map<String, Object> stats = generarEstadisticasVentas(ventas);

writer.println(" <resumen>");

writer.println(String.format(" <totalVentas>%,.2f</totalVentas>", stats.get("totalVentas")));

writer.println(String.format(" <cantidadVentas>%d</cantidadVentas>", ventas.size()));

writer.println(String.format(" <productosVendidos>%d</productosVendidos>", stats.get("totalProductos")));

writer.println(String.format(" <ticketPromedio>%,.2f</ticketPromedio>",

((double)stats.get("totalVentas"))/ventas.size()));

writer.println(" </resumen>");

writer.println(" <ventas>");

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");

for (Venta venta : ventas) {

writer.println(String.format(" <venta id=\"%d\" fecha=\"%s\" total=\"%.2f\">",

venta.getId(),

sdf.format(venta.getFecha()),

venta.getTotal()));

for (Producto producto : venta.getProductos()) {

writer.println(String.format(

" <producto nombre=\"%s\" cantidad=\"%d\" precioUnitario=\"%.2f\" total=\"%.2f\"/>",

producto.getNombre(),

producto.getCantidad(),

producto.getPrecioUnitario(),

producto.getPrecioUnitario() \* producto.getCantidad()

));

}

writer.println(" </venta>");

}

writer.println(" </ventas>");

} else {

writer.println(" <mensaje>Tipo de reporte no soportado para exportación XML</mensaje>");

}

writer.println("</reporte>");

}

}

private void exportarAHTML(String filename) throws Exception {

try (PrintWriter writer = new PrintWriter(new FileOutputStream(filename))) {

writer.println("<!DOCTYPE html>");

writer.println("<html>");

writer.println("<head>");

writer.println("<title>Reporte de " + tipoReporteActual + "</title>");

writer.println("<style>");

writer.println("body { font-family: Arial, sans-serif; margin: 20px; }");

writer.println("h1 { color: #2c3e50; }");

writer.println("table { width: 100%; border-collapse: collapse; }");

writer.println("th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }");

writer.println("th { background-color: #f2f2f2; }");

writer.println("</style>");

writer.println("</head>");

writer.println("<body>");

writer.println("<h1>Reporte de " + tipoReporteActual + "</h1>");

writer.println("<p>Generado: " + new Date() + "</p>");

writer.println("<p>Rango: " + panelVentas.getFechaInicio() + " - " + panelVentas.getFechaFin() + "</p>");

if ("VENTAS".equals(tipoReporteActual)) {

List<Venta> ventas = obtenerVentasDesdeBD(panelVentas.getFechaInicio(), panelVentas.getFechaFin());

// Tabla de resumen

writer.println("<h2>Resumen de Ventas</h2>");

writer.println("<table>");

writer.println("<tr><th>Total Ventas</th><th>Cantidad Ventas</th><th>Productos Vendidos</th><th>Ticket Promedio</th></tr>");

Map<String, Object> stats = generarEstadisticasVentas(ventas);

writer.println(String.format("<tr><td>$%,.2f</td><td>%d</td><td>%d</td><td>$%,.2f</td></tr>",

stats.get("totalVentas"),

ventas.size(),

stats.get("totalProductos"),

((double)stats.get("totalVentas"))/ventas.size()));

writer.println("</table>");

// Tabla de detalle

writer.println("<h2>Detalle de Ventas</h2>");

writer.println("<table>");

writer.println("<tr><th>ID Venta</th><th>Fecha</th><th>Producto</th><th>Cantidad</th><th>Precio Unitario</th><th>Total</th></tr>");

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (Venta venta : ventas) {

for (Producto producto : venta.getProductos()) {

writer.println(String.format(

"<tr><td>%d</td><td>%s</td><td>%s</td><td>%d</td><td>$%.2f</td><td>$%.2f</td></tr>",

venta.getId(),

sdf.format(venta.getFecha()),

producto.getNombre(),

producto.getCantidad(),

producto.getPrecioUnitario(),

producto.getPrecioUnitario() \* producto.getCantidad()

));

}

}

writer.println("</table>");

} else {

writer.println("<p>Tipo de reporte no soportado para exportación HTML</p>");

}

writer.println("</body>");

writer.println("</html>");

}

}

private void exportarAPDF(String filename) throws Exception {

pdfExporter.exportarAPDF(

filename,

tipoReporteActual,

panelVentas.getFechaInicio(),

panelVentas.getFechaFin()

);

}

public void agregarReporteVentas(Document document, List<Venta> ventas, PdfWriter writer) throws DocumentException {

// Resumen estadístico

Map<String, Object> stats = generarEstadisticasVentas(ventas);

PdfPTable summaryTable = new PdfPTable(4);

summaryTable.setWidthPercentage(100);

summaryTable.setSpacingBefore(10f);

summaryTable.setSpacingAfter(10f);

// Encabezado de la tabla de resumen

addSummaryHeaderCell(summaryTable, "Total Ventas");

addSummaryHeaderCell(summaryTable, "Cantidad Ventas");

addSummaryHeaderCell(summaryTable, "Productos Vendidos");

addSummaryHeaderCell(summaryTable, "Ticket Promedio");

// Datos del resumen

addSummaryDataCell(summaryTable, String.format("$%,.2f", stats.get("totalVentas")));

addSummaryDataCell(summaryTable, String.valueOf(ventas.size()));

addSummaryDataCell(summaryTable, String.valueOf(stats.get("totalProductos")));

addSummaryDataCell(summaryTable, String.format("$%,.2f", ((double)stats.get("totalVentas"))/ventas.size()));

document.add(summaryTable);

// Tabla de detalle de ventas

Paragraph detalleTitle = new Paragraph("Detalle de Ventas",

new Font(Font.FontFamily.HELVETICA, 12, Font.BOLD));

detalleTitle.setSpacingBefore(15f);

document.add(detalleTitle);

PdfPTable detailTable = new PdfPTable(6);

detailTable.setWidthPercentage(100);

detailTable.setSpacingBefore(5f);

// Configurar anchos de columnas

float[] columnWidths = {1f, 2f, 3f, 1f, 1.5f, 1.5f};

detailTable.setWidths(columnWidths);

// Encabezados de la tabla de detalle

addDetailHeaderCell(detailTable, "ID Venta");

addDetailHeaderCell(detailTable, "Fecha");

addDetailHeaderCell(detailTable, "Producto");

addDetailHeaderCell(detailTable, "Cantidad");

addDetailHeaderCell(detailTable, "Precio Unitario");

addDetailHeaderCell(detailTable, "Total");

// Llenar datos

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (Venta venta : ventas) {

for (Producto producto : venta.getProductos()) {

addDetailDataCell(detailTable, String.valueOf(venta.getId()));

addDetailDataCell(detailTable, sdf.format(venta.getFecha()));

addDetailDataCell(detailTable, producto.getNombre());

addDetailDataCell(detailTable, String.valueOf(producto.getCantidad()));

addDetailDataCell(detailTable, String.format("$%.2f", producto.getPrecioUnitario()));

addDetailDataCell(detailTable, String.format("$%.2f",

producto.getPrecioUnitario() \* producto.getCantidad()));

}

}

document.add(detailTable);

// Pie de página

Paragraph footer = new Paragraph("Fin del reporte",

new Font(Font.FontFamily.HELVETICA, 10, Font.ITALIC));

footer.setSpacingBefore(20f);

footer.setAlignment(Element.ALIGN\_CENTER);

document.add(footer);

}

private void addSummaryHeaderCell(PdfPTable table, String text) {

PdfPCell cell = new PdfPCell(new Phrase(text,

new Font(Font.FontFamily.HELVETICA, 10, Font.BOLD)));

cell.setHorizontalAlignment(Element.ALIGN\_CENTER);

cell.setBackgroundColor(new BaseColor(220, 220, 220));

cell.setPadding(5f);

table.addCell(cell);

}

private void addSummaryDataCell(PdfPTable table, String text) {

PdfPCell cell = new PdfPCell(new Phrase(text,

new Font(Font.FontFamily.HELVETICA, 10)));

cell.setHorizontalAlignment(Element.ALIGN\_CENTER);

cell.setPadding(5f);

table.addCell(cell);

}

private void addDetailHeaderCell(PdfPTable table, String text) {

PdfPCell cell = new PdfPCell(new Phrase(text,

new Font(Font.FontFamily.HELVETICA, 8, Font.BOLD)));

cell.setHorizontalAlignment(Element.ALIGN\_CENTER);

cell.setBackgroundColor(new BaseColor(220, 220, 220));

cell.setPadding(3f);

table.addCell(cell);

}

private void addDetailDataCell(PdfPTable table, String text) {

PdfPCell cell = new PdfPCell(new Phrase(text,

new Font(Font.FontFamily.HELVETICA, 8)));

cell.setPadding(3f);

table.addCell(cell);

}

public void agregarEncabezadoReporte(Document document, String tipoReporte, Date fechaInicio, Date fechaFin) throws DocumentException {

// Si no se proporcionan fechas, usar null

if (fechaInicio == null && fechaFin == null) {

// Lógica del encabezado sin fechas

} else {

// Crear tabla para el encabezado

PdfPTable headerTable = new PdfPTable(2);

headerTable.setWidthPercentage(100);

headerTable.setWidths(new int[]{1, 3});

// Logo (si está disponible)

try {

Image logo = Image.getInstance("imagen/logo.png");

logo.scaleToFit(100, 100);

PdfPCell logoCell = new PdfPCell(logo);

logoCell.setBorder(Rectangle.NO\_BORDER);

headerTable.addCell(logoCell);

} catch (Exception e) {

PdfPCell emptyCell = new PdfPCell(new Phrase(""));

emptyCell.setBorder(Rectangle.NO\_BORDER);

headerTable.addCell(emptyCell);

}

// Información de la empresa

PdfPCell infoCell = new PdfPCell();

infoCell.setBorder(Rectangle.NO\_BORDER);

Paragraph empresa = new Paragraph("El Habanerito",

new Font(Font.FontFamily.HELVETICA, 18, Font.BOLD));

empresa.setAlignment(Element.ALIGN\_RIGHT);

Paragraph reporte = new Paragraph("Reporte de " + tipoReporteActual,

new Font(Font.FontFamily.HELVETICA, 14, Font.BOLD));

reporte.setAlignment(Element.ALIGN\_RIGHT);

Paragraph fecha = new Paragraph("Generado: " + new SimpleDateFormat("dd/MM/yyyy HH:mm").format(new Date()),

new Font(Font.FontFamily.HELVETICA, 10));

fecha.setAlignment(Element.ALIGN\_RIGHT);

infoCell.addElement(empresa);

infoCell.addElement(reporte);

infoCell.addElement(fecha);

headerTable.addCell(infoCell);

document.add(headerTable);

// Línea separadora

Paragraph separator = new Paragraph();

separator.add(new Chunk(new Chunk()));

document.add(separator);

// Información del rango de fechas

if (panelVentas.getFechaInicio() != null || panelVentas.getFechaFin() != null) {

Paragraph rangoFechas = new Paragraph();

rangoFechas.add("Período: ");

if (panelVentas.getFechaInicio() != null) {

rangoFechas.add(new SimpleDateFormat("dd/MM/yyyy").format(panelVentas.getFechaInicio()));

}

rangoFechas.add(" - ");

if (panelVentas.getFechaFin() != null) {

rangoFechas.add(new SimpleDateFormat("dd/MM/yyyy").format(panelVentas.getFechaFin()));

}

document.add(rangoFechas);

}

document.add(Chunk.NEWLINE);}

}

public void imprimirReporte() {

exportarReporte(); // usamos exportar como impresión

}

private void abrirArchivoGenerado(String filename) {

File archivo = new File("C:\\Users\\Anahi\\eclipse-workspace\\punto\_venta\_2\\Reportes\\" +

tipoReporteActual + "\\" + filename);

// Si no se encuentra en la subcarpeta, buscar en la carpeta principal

if (!archivo.exists()) {

archivo = new File("C:\\Users\\Anahi\\eclipse-workspace\\punto\_venta\_2\\Reportes\\" + filename);

}

if (!archivo.exists()) {

mostrarErrorEnPanel("No se encontró el archivo generado: " + archivo.getAbsolutePath());

return;

}

if (!verificarIntegridadArchivo(archivo)) {

mostrarErrorEnPanel("El archivo está dañado o no es válido: " + archivo.getName());

return;

}

try {

if (Desktop.isDesktopSupported()) {

// Verificar que el archivo no esté bloqueado

if (!archivo.canRead()) {

mostrarErrorEnPanel("El archivo está siendo usado por otro proceso. Cierre otros programas que puedan estar usándolo.");

return;

}

// Abrir con la aplicación predeterminada

Desktop.getDesktop().open(archivo);

// Mostrar mensaje de éxito

mostrarMensajeEnPanel("Documento abierto exitosamente:\n" + archivo.getAbsolutePath());

} else {

mostrarErrorEnPanel("No se puede abrir el archivo automáticamente en este sistema.");

}

} catch (IOException e) {

mostrarErrorEnPanel("Error al abrir el archivo: " + e.getMessage());

e.printStackTrace();

}

}

public void onEndPage(PdfWriter writer, Document document) {

try {

agregarNumeroPagina(writer, document);

} catch (DocumentException e) {

panelVentas.mostrarError("Error al agregar número de página: " + e.getMessage());

}

}

private void agregarNumeroPagina(PdfWriter writer, Document document) throws DocumentException {

PdfContentByte cb = writer.getDirectContent();

Phrase footer = new Phrase(

String.format("Página %d", writer.getPageNumber()),

new Font(Font.FontFamily.HELVETICA, 8)

);

ColumnText.showTextAligned(

cb,

Element.ALIGN\_CENTER,

footer,

(document.right() - document.left()) / 2 + document.leftMargin(),

document.bottom() - 15,

0

);

}

public void agregarGraficos(Document document, List<Venta> ventas, PdfWriter writer)

throws DocumentException, IOException {

if (ventas == null || ventas.isEmpty()) {

return;

}

// 1. Gráfico de métodos de pago (Pie Chart)

DefaultPieDataset paymentDataset = new DefaultPieDataset();

Map<String, Double> metodosPago = ventas.stream()

.filter(v -> v.getMetodoPago() != null)

.collect(Collectors.groupingBy(

Venta::getMetodoPago,

Collectors.summingDouble(Venta::getTotal)

) );

metodosPago.forEach(paymentDataset::setValue);

JFreeChart paymentChart = ChartFactory.createPieChart(

"Distribución por Método de Pago",

paymentDataset,

true, true, false);

// 2. Gráfico de productos más vendidos (Bar Chart)

DefaultCategoryDataset productDataset = new DefaultCategoryDataset();

Map<String, Long> productosVendidos = ventas.stream()

.flatMap(v -> v.getProductos().stream())

.collect(Collectors.groupingBy(

Producto::getNombre,

Collectors.summingLong(Producto::getCantidad)

));

// Ordenar y tomar los top 10 productos

productosVendidos.entrySet().stream()

.sorted(Map.Entry.<String, Long>comparingByValue().reversed())

.limit(10)

.forEach(e -> productDataset.addValue(e.getValue(), "Ventas", e.getKey()));

JFreeChart productChart = ChartFactory.createBarChart(

"Top 10 Productos Más Vendidos",

"Productos",

"Cantidad Vendida",

productDataset,

PlotOrientation.VERTICAL,

true, true, false);

// Configuración de tamaño y calidad

int width = 500;

int height = 300;

float quality = 1.0f;

// Agregar gráficos al documento

Paragraph chartTitle = new Paragraph("Análisis Gráfico",

new Font(Font.FontFamily.HELVETICA, 14, Font.BOLD));

chartTitle.setSpacingBefore(20f);

document.add(chartTitle);

// Gráfico de métodos de pago

BufferedImage paymentImage = paymentChart.createBufferedImage(width, height);

Image paymentPdfImage = Image.getInstance(writer, paymentImage, quality);

paymentPdfImage.setAlignment(Image.MIDDLE);

document.add(paymentPdfImage);

// Espacio entre gráficos

document.add(Chunk.NEWLINE);

// Gráfico de productos

BufferedImage productImage = productChart.createBufferedImage(width, height);

Image productPdfImage = Image.getInstance(writer, productImage, quality);

productPdfImage.setAlignment(Image.MIDDLE);

document.add(productPdfImage);

}

public void cargarDatosInventario() {

try {

List<Producto> productos = inventarioDAO.obtenerTodosProductos();

actualizarVistaInventario(productos);

} catch (Exception e) {

mostrarErrorEnPanel("Error al cargar inventario: " + e.getMessage());

}

}

private void actualizarVistaInventario(List<Producto> productos) {

System.out.println("[UI] Actualizando vista con " + productos.size() + " productos");

if (panelInventario == null) {

System.err.println("[ERROR] No se puede actualizar vista: panelInventario es null");

return;

}

// Calcular estadísticas

int total = productos.size();

int bajoStock = (int) productos.stream()

.filter(p -> p.getCantidadDisponible() <= p.getStockMinimo())

.count();

int sinStock = (int) productos.stream()

.filter(p -> p.getCantidadDisponible() == 0)

.count();

System.out.println("[UI] Estadísticas - Total: " + total +

", Bajo stock: " + bajoStock +

", Sin stock: " + sinStock);

// Calcular promedios de stock

double stockPromedio = productos.stream()

.mapToInt(Producto::getCantidadDisponible)

.average()

.orElse(0);

double stockMinPromedio = productos.stream()

.mapToInt(Producto::getStockMinimo)

.average()

.orElse(0);

double stockMaxPromedio = productos.stream()

.mapToInt(Producto::getStockMaximo)

.average()

.orElse(0);

System.out.println("[UI] Promedios - Stock: " + stockPromedio +

", Mín: " + stockMinPromedio +

", Máx: " + stockMaxPromedio);

// Distribución por categoría

Map<String, Integer> distribucionCategorias = productos.stream()

.collect(Collectors.groupingBy(

Producto::getCategoria,

Collectors.summingInt(p -> 1)

));

System.out.println("[UI] Distribución por categoría: " + distribucionCategorias);

// Actualizar la vista en el hilo de EDT

SwingUtilities.invokeLater(() -> {

System.out.println("[UI] Actualizando componentes en EDT...");

try {

panelInventario.actualizarTablaInventario(productos);

panelInventario.actualizarEstadisticas(total, bajoStock, sinStock);

panelInventario.actualizarGraficas(stockPromedio, stockMinPromedio, stockMaxPromedio, distribucionCategorias);

System.out.println("[UI] Componentes actualizados correctamente");

} catch (Exception e) {

System.err.println("[UI] Error al actualizar componentes: " + e.getMessage());

e.printStackTrace();

}

});

}

public void filtrarInventario(String filtro) {

System.out.println("[DEBUG] Filtrando inventario por: " + filtro);

try {

List<Producto> productosFiltrados;

switch (filtro) {

case "Bajo Stock":

productosFiltrados = inventarioDAO.buscarProductosNecesitanReposicion();

System.out.println("[DEBUG] Productos bajo stock encontrados: " + productosFiltrados.size());

break;

case "Sobre Stock":

productosFiltrados = inventarioDAO.buscarProductosConExcesoStock();

System.out.println("[DEBUG] Productos sobre stock encontrados: " + productosFiltrados.size());

break;

case "Categoría":

String categoria = JOptionPane.showInputDialog("Ingrese la categoría:");

if (categoria == null || categoria.isEmpty()) {

System.out.println("[DEBUG] Filtro por categoría cancelado");

return;

}

productosFiltrados = inventarioDAO.buscarPorCategoria(categoria);

System.out.println("[DEBUG] Productos por categoría '" + categoria + "' encontrados: " + productosFiltrados.size());

break;

case "Proveedor":

String proveedor = JOptionPane.showInputDialog("Ingrese el proveedor:");

if (proveedor == null || proveedor.isEmpty()) {

System.out.println("[DEBUG] Filtro por proveedor cancelado");

return;

}

productosFiltrados = inventarioDAO.buscarPorProveedor(proveedor);

System.out.println("[DEBUG] Productos por proveedor '" + proveedor + "' encontrados: " + productosFiltrados.size());

break;

default: // "Todos"

productosFiltrados = inventarioDAO.obtenerTodosProductos();

System.out.println("[DEBUG] Todos los productos obtenidos: " + productosFiltrados.size());

break;

}

if (panelInventario == null) {

System.err.println("[ERROR] Panel de inventario no está inicializado");

return;

}

actualizarVistaInventario(productosFiltrados);

} catch (Exception e) {

System.err.println("[ERROR] Al filtrar inventario: " + e.getMessage());

e.printStackTrace();

mostrarErrorEnPanel("Error al filtrar inventario: " + e.getMessage());

}

}

public void exportarReporteInventario(String formato) {

try {

List<Producto> productos = inventarioDAO.obtenerTodosProductos();

// Calcular estadísticas para el reporte

int totalProductos = productos.size();

int bajoStock = (int) productos.stream()

.filter(p -> p.getCantidadDisponible() <= p.getStockMinimo())

.count();

int sinStock = (int) productos.stream()

.filter(p -> p.getCantidadDisponible() == 0)

.count();

double stockPromedio = productos.stream()

.mapToInt(Producto::getCantidadDisponible)

.average()

.orElse(0);

Map<String, Integer> distribucionCategorias = productos.stream()

.collect(Collectors.groupingBy(

Producto::getCategoria,

Collectors.summingInt(p -> 1)

));

// Seleccionar el método de exportación según el formato

switch (formato.toUpperCase()) {

case "PDF":

exportarInventarioAPDF(productos, totalProductos, bajoStock, sinStock,

stockPromedio, distribucionCategorias);

break;

case "EXCEL":

exportarInventarioAExcel(productos, totalProductos, bajoStock, sinStock,

stockPromedio, distribucionCategorias);

break;

case "XML":

exportarInventarioAHTML(productos, totalProductos, bajoStock, sinStock,

stockPromedio, distribucionCategorias);

break;

default:

mostrarErrorEnPanel("Formato no soportado: " + formato);

}

} catch (Exception e) {

mostrarErrorEnPanel("Error al exportar reporte: " + e.getMessage());

e.printStackTrace();

}

}

private void exportarInventarioAPDF(List<Producto> productos, int totalProductos, int bajoStock,

int sinStock, double stockPromedio,

Map<String, Integer> distribucionCategorias) throws Exception {

String filename = "reporte\_inventario\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".pdf";

Document document = new Document(PageSize.A4.rotate());

PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(filename));

document.open();

// Encabezado

agregarEncabezadoReporte(document, "INVENTARIO", null, null);

// Resumen estadístico

PdfPTable summaryTable = new PdfPTable(4);

summaryTable.setWidthPercentage(100);

summaryTable.setSpacingBefore(10f);

addSummaryHeaderCell(summaryTable, "Total Productos");

addSummaryHeaderCell(summaryTable, "Bajo Stock");

addSummaryHeaderCell(summaryTable, "Sin Stock");

addSummaryHeaderCell(summaryTable, "Stock Promedio");

addSummaryDataCell(summaryTable, String.valueOf(totalProductos));

addSummaryDataCell(summaryTable, String.valueOf(bajoStock));

addSummaryDataCell(summaryTable, String.valueOf(sinStock));

addSummaryDataCell(summaryTable, String.format("%.2f", stockPromedio));

document.add(summaryTable);

// Tabla de detalle

Paragraph detalleTitle = new Paragraph("Detalle de Inventario",

new Font(Font.FontFamily.HELVETICA, 12, Font.BOLD));

detalleTitle.setSpacingBefore(15f);

document.add(detalleTitle);

PdfPTable table = new PdfPTable(8);

table.setWidthPercentage(100);

table.setWidths(new float[]{1f, 3f, 2f, 1f, 1f, 1f, 1.5f, 2f});

// Encabezados

addDetailHeaderCell(table, "ID");

addDetailHeaderCell(table, "Producto");

addDetailHeaderCell(table, "Categoría");

addDetailHeaderCell(table, "Stock");

addDetailHeaderCell(table, "Mín");

addDetailHeaderCell(table, "Máx");

addDetailHeaderCell(table, "Precio");

addDetailHeaderCell(table, "Proveedor");

// Datos

for (Producto producto : productos) {

addDetailDataCell(table, producto.getId());

addDetailDataCell(table, producto.getNombre());

addDetailDataCell(table, producto.getCategoria());

addDetailDataCell(table, String.valueOf(producto.getCantidadDisponible()));

addDetailDataCell(table, String.valueOf(producto.getStockMinimo()));

addDetailDataCell(table, String.valueOf(producto.getStockMaximo()));

addDetailDataCell(table, String.format("$%.2f", producto.getPrecioVenta()));

addDetailDataCell(table, producto.getProveedor());

}

document.add(table);

// Gráficas

agregarGraficasInventario(document, writer, stockPromedio, distribucionCategorias);

document.close();

// Abrir el archivo generado

abrirArchivoGenerado(filename);

}

private void agregarGraficasInventario(Document document, PdfWriter writer,

double stockPromedio,

Map<String, Integer> distribucionCategorias)

throws IOException, DocumentException {

if (distribucionCategorias.isEmpty()) return;

// 1. Gráfico de niveles de stock

DefaultCategoryDataset datasetStock = new DefaultCategoryDataset();

datasetStock.addValue(stockPromedio, "Stock", "Actual");

JFreeChart chartStock = ChartFactory.createBarChart(

"Nivel de Stock Promedio",

"",

"Cantidad",

datasetStock

);

// 2. Gráfico de distribución por categoría

DefaultPieDataset datasetCategorias = new DefaultPieDataset();

distribucionCategorias.forEach(datasetCategorias::setValue);

JFreeChart chartCategorias = ChartFactory.createPieChart(

"Distribución por Categoría",

datasetCategorias,

true, true, false

);

// Configuración de tamaño y calidad

int width = 500;

int height = 300;

float quality = 1.0f;

// Agregar gráficos al documento

Paragraph chartTitle = new Paragraph("Análisis Gráfico",

new Font(Font.FontFamily.HELVETICA, 14, Font.BOLD));

chartTitle.setSpacingBefore(20f);

document.add(chartTitle);

// Gráfico de stock

BufferedImage stockImage = chartStock.createBufferedImage(width, height);

Image stockPdfImage = Image.getInstance(writer, stockImage, quality);

stockPdfImage.setAlignment(Image.MIDDLE);

document.add(stockPdfImage);

// Espacio entre gráficos

document.add(Chunk.NEWLINE);

// Gráfico de categorías

BufferedImage categoriasImage = chartCategorias.createBufferedImage(width, height);

Image categoriasPdfImage = Image.getInstance(writer, categoriasImage, quality);

categoriasPdfImage.setAlignment(Image.MIDDLE);

document.add(categoriasPdfImage);

}

private void exportarInventarioAExcel(List<Producto> productos, int totalProductos, int bajoStock,

int sinStock, double stockPromedio,

Map<String, Integer> distribucionCategorias) throws Exception {

String filename = "reporte\_inventario\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".xlsx";

try (Workbook workbook = new XSSFWorkbook()) {

Sheet sheet = workbook.createSheet("Inventario");

// Estilos

CellStyle headerStyle = crearEstiloEncabezado(workbook);

CellStyle titleStyle = crearEstiloTitulo(workbook);

CellStyle currencyStyle = crearEstiloMoneda(workbook);

int currentRow = 0;

// Título

Row titleRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, titleRow, 0, 7, "REPORTE DE INVENTARIO", titleStyle);

// Fecha

Row dateRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, dateRow, 0, 7,

"Generado: " + new SimpleDateFormat("yyyy-MM-dd HH:mm").format(new Date()), null);

currentRow++;

// Resumen estadístico

Row statsTitleRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, statsTitleRow, 0, 7, "RESUMEN ESTADÍSTICO", titleStyle);

Row statsHeaderRow = sheet.createRow(currentRow++);

crearCelda(statsHeaderRow, 0, "Total Productos", headerStyle);

crearCelda(statsHeaderRow, 1, "Bajo Stock", headerStyle);

crearCelda(statsHeaderRow, 2, "Sin Stock", headerStyle);

crearCelda(statsHeaderRow, 3, "Stock Promedio", headerStyle);

Row statsDataRow = sheet.createRow(currentRow++);

crearCelda(statsDataRow, 0, totalProductos, null);

crearCelda(statsDataRow, 1, bajoStock, null);

crearCelda(statsDataRow, 2, sinStock, null);

crearCelda(statsDataRow, 3, stockPromedio, null);

currentRow++;

// Encabezados de la tabla

Row headerRow = sheet.createRow(currentRow++);

String[] headers = {"ID", "Producto", "Categoría", "Stock", "Stock Mín", "Stock Máx", "Precio", "Proveedor"};

for (int i = 0; i < headers.length; i++) {

crearCelda(headerRow, i, headers[i], headerStyle);

}

// Datos de productos

for (Producto producto : productos) {

Row row = sheet.createRow(currentRow++);

crearCelda(row, 0, producto.getId(), null);

crearCelda(row, 1, producto.getNombre(), null);

crearCelda(row, 2, producto.getCategoria(), null);

crearCelda(row, 3, producto.getCantidadDisponible(), null);

crearCelda(row, 4, producto.getStockMinimo(), null);

crearCelda(row, 5, producto.getStockMaximo(), null);

crearCelda(row, 6, producto.getPrecioVenta(), currencyStyle);

crearCelda(row, 7, producto.getProveedor(), null);

}

// Autoajustar columnas

for (int i = 0; i < headers.length; i++) {

sheet.autoSizeColumn(i);

}

// Guardar archivo

try (FileOutputStream outputStream = new FileOutputStream(filename)) {

workbook.write(outputStream);

}

}

abrirArchivoGenerado(filename);

}

private void exportarInventarioAHTML(List<Producto> productos, int totalProductos, int bajoStock,

int sinStock, double stockPromedio,

Map<String, Integer> distribucionCategorias) throws Exception {

String filename = "reporte\_inventario\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".html";

try (PrintWriter writer = new PrintWriter(new FileOutputStream(filename))) {

writer.println("<!DOCTYPE html>");

writer.println("<html lang='es'>");

writer.println("<head>");

writer.println("<meta charset='UTF-8'>");

writer.println("<title>Reporte de Inventario</title>");

writer.println("<style>");

writer.println("body { font-family: Arial, sans-serif; margin: 20px; }");

writer.println("h1 { color: #2c3e50; }");

writer.println("table { width: 100%; border-collapse: collapse; margin-bottom: 20px; }");

writer.println("th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }");

writer.println("th { background-color: #f2f2f2; }");

writer.println(".resumen { background-color: #f8f9fa; padding: 15px; margin-bottom: 20px; }");

writer.println("</style>");

writer.println("</head>");

writer.println("<body>");

writer.println("<h1>Reporte de Inventario</h1>");

writer.println("<div class='resumen'>");

writer.println("<p><strong>Generado:</strong> " + new Date() + "</p>");

writer.println("<p><strong>Total Productos:</strong> " + totalProductos + "</p>");

writer.println("<p><strong>Productos bajo stock:</strong> " + bajoStock + "</p>");

writer.println("<p><strong>Productos sin stock:</strong> " + sinStock + "</p>");

writer.println("<p><strong>Stock promedio:</strong> " + String.format("%.2f", stockPromedio) + "</p>");

writer.println("</div>");

writer.println("<h2>Detalle de Productos</h2>");

writer.println("<table>");

writer.println("<tr><th>ID</th><th>Producto</th><th>Categoría</th><th>Stock</th><th>Stock Mín</th><th>Stock Máx</th><th>Precio</th><th>Proveedor</th></tr>");

for (Producto producto : productos) {

writer.println(String.format(

"<tr><td>%s</td><td>%s</td><td>%s</td><td>%d</td><td>%d</td><td>%d</td><td>$%.2f</td><td>%s</td></tr>",

producto.getId(),

producto.getNombre(),

producto.getCategoria(),

producto.getCantidadDisponible(),

producto.getStockMinimo(),

producto.getStockMaximo(),

producto.getPrecioVenta(),

producto.getProveedor()

));

}

writer.println("</table>");

writer.println("<h2>Distribución por Categoría</h2>");

writer.println("<ul>");

distribucionCategorias.forEach((categoria, cantidad) -> {

writer.println(String.format("<li><strong>%s:</strong> %d productos (%.1f%%)</li>",

categoria,

cantidad,

(cantidad \* 100.0 / totalProductos)

));

});

writer.println("</ul>");

writer.println("</body>");

writer.println("</html>");

}

abrirArchivoGenerado(filename);

}

public void cargarDatosClientes() {

try {

List<Cliente> clientes = clienteDAO.obtenerTodos();

actualizarVistaClientes(clientes);

} catch (Exception e) {

mostrarErrorEnPanel("Error al cargar clientes: " + e.getMessage());

}

}

private void actualizarVistaClientes(List<Cliente> clientes) {

System.out.println("[UI] Actualizando vista con " + clientes.size() + " clientes");

if (panelClientes == null) {

System.err.println("[ERROR] No se puede actualizar vista: panelClientes es null");

return;

}

// Calcular estadísticas

int total = clientes.size();

int activos = (int) clientes.stream()

.filter(c -> c.getFechaEliminacion() == null)

.count();

int inactivos = total - activos;

// Registros por mes

Map<String, Integer> registrosPorMes = clientes.stream()

.collect(Collectors.groupingBy(

c -> new SimpleDateFormat("MMM").format(c.getFechaRegistro()),

Collectors.summingInt(c -> 1)

));

// Distribución de puntos

Map<String, Integer> distribucionPuntos = Map.of(

"0-100 pts", (int) clientes.stream().filter(c -> c.getPuntos() <= 100).count(),

"101-500 pts", (int) clientes.stream().filter(c -> c.getPuntos() > 100 && c.getPuntos() <= 500).count(),

"501+ pts", (int) clientes.stream().filter(c -> c.getPuntos() > 500).count()

);

// Actualizar la vista en el hilo de EDT

SwingUtilities.invokeLater(() -> {

try {

panelClientes.actualizarTablaClientes(clientes);

panelClientes.actualizarEstadisticas(total, activos, inactivos);

panelClientes.actualizarGraficas(registrosPorMes, distribucionPuntos);

} catch (Exception e) {

System.err.println("[UI] Error al actualizar componentes: " + e.getMessage());

e.printStackTrace();

}

});

}

public void filtrarClientes(String filtro) {

System.out.println("[DEBUG] Filtrando clientes por: " + filtro);

try {

List<Cliente> clientesFiltrados;

switch (filtro) {

case "Activos":

clientesFiltrados = clienteDAO.obtenerTodos().stream()

.filter(c -> c.getFechaEliminacion() == null)

.collect(Collectors.toList());

break;

case "Inactivos":

clientesFiltrados = clienteDAO.obtenerTodos().stream()

.filter(c -> c.getFechaEliminacion() != null)

.collect(Collectors.toList());

break;

case "Con Puntos":

clientesFiltrados = clienteDAO.obtenerTodos().stream()

.filter(c -> c.getPuntos() > 0)

.collect(Collectors.toList());

break;

case "Sin Puntos":

clientesFiltrados = clienteDAO.obtenerTodos().stream()

.filter(c -> c.getPuntos() == 0)

.collect(Collectors.toList());

break;

default: // "Todos"

clientesFiltrados = clienteDAO.obtenerTodos();

break;

}

actualizarVistaClientes(clientesFiltrados);

} catch (Exception e) {

mostrarErrorEnPanel("Error al filtrar clientes: " + e.getMessage());

}

}

public void exportarReporteClientes(String formato) {

try {

List<Cliente> clientes = clienteDAO.obtenerTodos();

// Calcular estadísticas para el reporte

int totalClientes = clientes.size();

int activos = (int) clientes.stream()

.filter(c -> c.getFechaEliminacion() == null)

.count();

Map<String, Integer> registrosPorMes = clientes.stream()

.collect(Collectors.groupingBy(

c -> new SimpleDateFormat("MMM").format(c.getFechaRegistro()),

Collectors.summingInt(c -> 1)

));

// Seleccionar el método de exportación según el formato

switch (formato.toUpperCase()) {

case "PDF":

exportarClientesAPDF(clientes, totalClientes, activos, registrosPorMes);

break;

case "EXCEL":

exportarClientesAExcel(clientes, totalClientes, activos, registrosPorMes);

break;

case "HTML":

exportarClientesAHTML(clientes, totalClientes, activos, registrosPorMes);

break;

default:

mostrarErrorEnPanel("Formato no soportado: " + formato);

}

} catch (Exception e) {

mostrarErrorEnPanel("Error al exportar reporte: " + e.getMessage());

}

}

private void exportarClientesAPDF(List<Cliente> clientes, int totalClientes, int activos,

Map<String, Integer> registrosPorMes) throws Exception {

String filename = "reporte\_clientes\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".pdf";

Document document = new Document(PageSize.A4.rotate());

PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(filename));

document.open();

// Encabezado

agregarEncabezadoReporte(document, "CLIENTES", null, null);

// Resumen estadístico

PdfPTable summaryTable = new PdfPTable(3);

summaryTable.setWidthPercentage(100);

summaryTable.setSpacingBefore(10f);

addSummaryHeaderCell(summaryTable, "Total Clientes");

addSummaryHeaderCell(summaryTable, "Clientes Activos");

addSummaryHeaderCell(summaryTable, "Clientes Inactivos");

addSummaryDataCell(summaryTable, String.valueOf(totalClientes));

addSummaryDataCell(summaryTable, String.valueOf(activos));

addSummaryDataCell(summaryTable, String.valueOf(totalClientes - activos));

document.add(summaryTable);

// Tabla de detalle

Paragraph detalleTitle = new Paragraph("Detalle de Clientes",

new Font(Font.FontFamily.HELVETICA, 12, Font.BOLD));

detalleTitle.setSpacingBefore(15f);

document.add(detalleTitle);

PdfPTable table = new PdfPTable(6);

table.setWidthPercentage(100);

table.setWidths(new float[]{1f, 1.5f, 3f, 2f, 1f, 2f});

// Encabezados

addDetailHeaderCell(table, "ID");

addDetailHeaderCell(table, "Teléfono");

addDetailHeaderCell(table, "Nombre");

addDetailHeaderCell(table, "Última Compra");

addDetailHeaderCell(table, "Puntos");

addDetailHeaderCell(table, "Fecha Registro");

// Datos

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

for (Cliente cliente : clientes) {

addDetailDataCell(table, cliente.getId());

addDetailDataCell(table, cliente.getTelefono());

addDetailDataCell(table, cliente.getNombre());

addDetailDataCell(table, cliente.getUltimaCompra() != null ? cliente.getUltimaCompra() : "N/A");

addDetailDataCell(table, String.valueOf(cliente.getPuntos()));

addDetailDataCell(table, cliente.getFechaRegistro() != null ? sdf.format(cliente.getFechaRegistro()) : "N/A");

}

document.add(table);

// Gráficas

agregarGraficasClientes(document, writer, registrosPorMes);

document.close();

// Abrir el archivo generado

abrirArchivoGenerado(filename);

}

private void agregarGraficasClientes(Document document, PdfWriter writer,

Map<String, Integer> registrosPorMes)

throws IOException, DocumentException {

if (registrosPorMes.isEmpty()) return;

// 1. Gráfico de registros por mes

DefaultCategoryDataset datasetRegistros = new DefaultCategoryDataset();

registrosPorMes.forEach((mes, cantidad) -> {

datasetRegistros.addValue(cantidad, "Registros", mes);

});

JFreeChart chartRegistros = ChartFactory.createBarChart(

"Registros de Clientes por Mes",

"Mes",

"Cantidad",

datasetRegistros

);

// 2. Gráfico de distribución de puntos (ejemplo)

DefaultPieDataset datasetPuntos = new DefaultPieDataset();

datasetPuntos.setValue("0-100 pts", 30);

datasetPuntos.setValue("101-500 pts", 45);

datasetPuntos.setValue("501+ pts", 25);

JFreeChart chartPuntos = ChartFactory.createPieChart(

"Distribución de Puntos",

datasetPuntos,

true, true, false

);

// Configuración de tamaño y calidad

int width = 500;

int height = 300;

float quality = 1.0f;

// Agregar gráficos al documento

Paragraph chartTitle = new Paragraph("Análisis Gráfico",

new Font(Font.FontFamily.HELVETICA, 14, Font.BOLD));

chartTitle.setSpacingBefore(20f);

document.add(chartTitle);

// Gráfico de registros

BufferedImage registrosImage = chartRegistros.createBufferedImage(width, height);

Image registrosPdfImage = Image.getInstance(writer, registrosImage, quality);

registrosPdfImage.setAlignment(Image.MIDDLE);

document.add(registrosPdfImage);

// Espacio entre gráficos

document.add(Chunk.NEWLINE);

// Gráfico de puntos

BufferedImage puntosImage = chartPuntos.createBufferedImage(width, height);

Image puntosPdfImage = Image.getInstance(writer, puntosImage, quality);

puntosPdfImage.setAlignment(Image.MIDDLE);

document.add(puntosPdfImage);

}

private void exportarClientesAHTML(List<Cliente> clientes, int totalClientes, int activos,

Map<String, Integer> registrosPorMes) throws Exception {

String filename = "reporte\_clientes\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".html";

try (PrintWriter writer = new PrintWriter(new FileOutputStream(filename))) {

writer.println("<!DOCTYPE html>");

writer.println("<html lang='es'>");

writer.println("<head>");

writer.println("<meta charset='UTF-8'>");

writer.println("<title>Reporte de Clientes</title>");

writer.println("<style>");

writer.println("body { font-family: Arial, sans-serif; margin: 20px; }");

writer.println("h1 { color: #2c3e50; }");

writer.println("table { width: 100%; border-collapse: collapse; margin-bottom: 20px; }");

writer.println("th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }");

writer.println("th { background-color: #f2f2f2; }");

writer.println(".resumen { background-color: #f8f9fa; padding: 15px; margin-bottom: 20px; }");

writer.println(".chart-container { display: flex; justify-content: space-around; margin: 20px 0; }");

writer.println(".chart { width: 45%; border: 1px solid #ddd; padding: 10px; }");

writer.println("</style>");

writer.println("</head>");

writer.println("<body>");

writer.println("<h1>Reporte de Clientes</h1>");

writer.println("<div class='resumen'>");

writer.println("<p><strong>Generado:</strong> " + new Date() + "</p>");

writer.println("<p><strong>Total Clientes:</strong> " + totalClientes + "</p>");

writer.println("<p><strong>Clientes activos:</strong> " + activos + "</p>");

writer.println("<p><strong>Clientes inactivos:</strong> " + (totalClientes - activos) + "</p>");

writer.println("</div>");

writer.println("<h2>Detalle de Clientes</h2>");

writer.println("<table>");

writer.println("<tr><th>ID</th><th>Teléfono</th><th>Nombre</th><th>Última Compra</th><th>Puntos</th><th>Fecha Registro</th></tr>");

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

for (Cliente cliente : clientes) {

writer.println(String.format(

"<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td><td>%d</td><td>%s</td></tr>",

cliente.getId(),

cliente.getTelefono(),

cliente.getNombre(),

cliente.getUltimaCompra() != null ? cliente.getUltimaCompra() : "N/A",

cliente.getPuntos(),

cliente.getFechaRegistro() != null ? sdf.format(cliente.getFechaRegistro()) : "N/A"

));

}

writer.println("</table>");

// Sección de gráficos (simulada con HTML básico)

writer.println("<div class='chart-container'>");

writer.println("<div class='chart'>");

writer.println("<h3>Registros por Mes</h3>");

writer.println("<ul>");

registrosPorMes.forEach((mes, cantidad) -> {

writer.println(String.format("<li><strong>%s:</strong> %d clientes</li>", mes, cantidad));

});

writer.println("</ul>");

writer.println("</div>");

writer.println("<div class='chart'>");

writer.println("<h3>Distribución de Puntos</h3>");

writer.println("<ul>");

writer.println("<li><strong>0-100 pts:</strong> " +

clientes.stream().filter(c -> c.getPuntos() <= 100).count() + " clientes</li>");

writer.println("<li><strong>101-500 pts:</strong> " +

clientes.stream().filter(c -> c.getPuntos() > 100 && c.getPuntos() <= 500).count() + " clientes</li>");

writer.println("<li><strong>501+ pts:</strong> " +

clientes.stream().filter(c -> c.getPuntos() > 500).count() + " clientes</li>");

writer.println("</ul>");

writer.println("</div>");

writer.println("</div>");

writer.println("</body>");

writer.println("</html>");

}

abrirArchivoGenerado(filename);

}

private void exportarClientesAExcel(List<Cliente> clientes, int totalClientes, int activos,

Map<String, Integer> registrosPorMes) throws Exception {

String filename = "reporte\_clientes\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".xlsx";

try (Workbook workbook = new XSSFWorkbook()) {

Sheet sheet = workbook.createSheet("Clientes");

// Estilos

CellStyle headerStyle = crearEstiloEncabezado(workbook);

CellStyle titleStyle = crearEstiloTitulo(workbook);

CellStyle dateStyle = crearEstiloFecha(workbook);

int currentRow = 0;

// Título

Row titleRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, titleRow, 0, 5, "REPORTE DE CLIENTES", titleStyle);

// Fecha

Row dateRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, dateRow, 0, 5,

"Generado: " + new SimpleDateFormat("yyyy-MM-dd HH:mm").format(new Date()), null);

currentRow++;

// Resumen estadístico

Row statsTitleRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, statsTitleRow, 0, 5, "RESUMEN ESTADÍSTICO", titleStyle);

Row statsHeaderRow = sheet.createRow(currentRow++);

crearCelda(statsHeaderRow, 0, "Total Clientes", headerStyle);

crearCelda(statsHeaderRow, 1, "Clientes Activos", headerStyle);

crearCelda(statsHeaderRow, 2, "Clientes Inactivos", headerStyle);

Row statsDataRow = sheet.createRow(currentRow++);

crearCelda(statsDataRow, 0, totalClientes, null);

crearCelda(statsDataRow, 1, activos, null);

crearCelda(statsDataRow, 2, totalClientes - activos, null);

currentRow++;

// Encabezados de la tabla

Row headerRow = sheet.createRow(currentRow++);

String[] headers = {"ID", "Teléfono", "Nombre", "Última Compra", "Puntos", "Fecha Registro"};

for (int i = 0; i < headers.length; i++) {

crearCelda(headerRow, i, headers[i], headerStyle);

}

// Datos de clientes

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

for (Cliente cliente : clientes) {

Row row = sheet.createRow(currentRow++);

crearCelda(row, 0, cliente.getId(), null);

crearCelda(row, 1, cliente.getTelefono(), null);

crearCelda(row, 2, cliente.getNombre(), null);

crearCelda(row, 3, cliente.getUltimaCompra() != null ? cliente.getUltimaCompra() : "N/A", null);

crearCelda(row, 4, cliente.getPuntos(), null);

if (cliente.getFechaRegistro() != null) {

Cell fechaCell = row.createCell(5);

fechaCell.setCellValue(cliente.getFechaRegistro());

fechaCell.setCellStyle(dateStyle);

} else {

crearCelda(row, 5, "N/A", null);

}

}

// Autoajustar columnas

for (int i = 0; i < headers.length; i++) {

sheet.autoSizeColumn(i);

}

// Hoja adicional para gráficos (requiere Apache POI 4.0+)

Sheet chartSheet = workbook.createSheet("Gráficos");

currentRow = 0;

// Datos para gráfico de registros por mes

Row chartTitleRow = chartSheet.createRow(currentRow++);

crearCeldaConMergedRegion(chartSheet, chartTitleRow, 0, 1, "Registros por Mes", titleStyle);

Row chartHeaderRow = chartSheet.createRow(currentRow++);

crearCelda(chartHeaderRow, 0, "Mes", headerStyle);

crearCelda(chartHeaderRow, 1, "Cantidad", headerStyle);

for (Map.Entry<String, Integer> entry : registrosPorMes.entrySet()) {

Row dataRow = chartSheet.createRow(currentRow++);

crearCelda(dataRow, 0, entry.getKey(), null);

crearCelda(dataRow, 1, entry.getValue(), null);

}

// Guardar archivo

try (FileOutputStream outputStream = new FileOutputStream(filename)) {

workbook.write(outputStream);

}

}

abrirArchivoGenerado(filename);

}

public Map<String, Object> generarEstadisticasClientes(List<Cliente> clientes) {

Map<String, Object> stats = new HashMap<>();

if (clientes == null || clientes.isEmpty()) {

return stats;

}

// Estadísticas básicas

int total = clientes.size();

int activos = (int) clientes.stream().filter(c -> c.getFechaEliminacion() == null).count();

int conPuntos = (int) clientes.stream().filter(c -> c.getPuntos() > 0).count();

// Registros por mes

Map<String, Integer> registrosPorMes = clientes.stream()

.collect(Collectors.groupingBy(

c -> new SimpleDateFormat("MMM yyyy").format(c.getFechaRegistro()),

Collectors.summingInt(c -> 1)

));

// Distribución de puntos

Map<String, Integer> distribucionPuntos = new LinkedHashMap<>();

distribucionPuntos.put("0-100 pts", (int) clientes.stream().filter(c -> c.getPuntos() <= 100).count());

distribucionPuntos.put("101-500 pts", (int) clientes.stream().filter(c -> c.getPuntos() > 100 && c.getPuntos() <= 500).count());

distribucionPuntos.put("501+ pts", (int) clientes.stream().filter(c -> c.getPuntos() > 500).count());

// Clientes con más puntos (Top 5)

List<Cliente> topClientes = clientes.stream()

.sorted((c1, c2) -> Integer.compare(c2.getPuntos(), c1.getPuntos()))

.limit(5)

.collect(Collectors.toList());

stats.put("totalClientes", total);

stats.put("clientesActivos", activos);

stats.put("clientesInactivos", total - activos);

stats.put("clientesConPuntos", conPuntos);

stats.put("registrosPorMes", registrosPorMes);

stats.put("distribucionPuntos", distribucionPuntos);

stats.put("topClientes", topClientes);

return stats;

}

public void actualizarGraficasClientes() {

List<Cliente> clientes = clienteDAO.obtenerTodos();

Map<String, Object> stats = generarEstadisticasClientes(clientes);

if (panelClientes != null) {

panelClientes.actualizarGraficas(

(Map<String, Integer>) stats.get("registrosPorMes"),

(Map<String, Integer>) stats.get("distribucionPuntos")

);

}

}

public List<Cliente> filtrarClientesPorRangoPuntos(int min, int max) {

return clienteDAO.obtenerTodos().stream()

.filter(c -> c.getPuntos() >= min && c.getPuntos() <= max)

.collect(Collectors.toList());

}

public List<Cliente> filtrarClientesPorPeriodo(Date fechaInicio, Date fechaFin) {

return clienteDAO.obtenerTodos().stream()

.filter(c -> c.getFechaRegistro() != null &&

!c.getFechaRegistro().before(fechaInicio) &&

!c.getFechaRegistro().after(fechaFin))

.collect(Collectors.toList());

}

public void cargarDatosProveedores() {

try {

if (proveedorDAO == null) {

proveedorDAO = new ProveedorDAO(); // Inicializa si es null

}

List<Proveedor> proveedores = proveedorDAO.obtenerTodosProveedores();

actualizarVistaProveedores(proveedores);

} catch (Exception e) {

mostrarErrorEnPanel("Error al cargar proveedores: " + e.getMessage());

}

}

private void actualizarVistaProveedores(List<Proveedor> proveedores) {

System.out.println("[UI] Actualizando vista con " + proveedores.size() + " proveedores");

if (panelProveedores == null) {

System.err.println("[ERROR] No se puede actualizar vista: panelProveedores es null");

return;

}

// Calcular estadísticas

int total = proveedores.size();

// Proveedores visitados este mes

Calendar cal = Calendar.getInstance();

cal.set(Calendar.DAY\_OF\_MONTH, 1);

Date inicioMes = new Date(cal.getTimeInMillis());

int visitadosEsteMes = (int) proveedores.stream()

.filter(p -> p.getUltimaVisita() != null &&

p.getUltimaVisita().after(inicioMes))

.count();

// Última visita registrada

String ultimaVisita = proveedores.stream()

.filter(p -> p.getUltimaVisita() != null)

.map(Proveedor::getUltimaVisita)

.max(Timestamp::compareTo)

.map(t -> new SimpleDateFormat("dd/MM/yyyy").format(t))

.orElse("N/A");

// Distribución por producto suministrado

Map<String, Integer> distribucionProductos = proveedores.stream()

.collect(Collectors.groupingBy(

p -> p.getProductoSuministrado() != null && !p.getProductoSuministrado().isEmpty() ?

p.getProductoSuministrado() : "No especificado",

Collectors.summingInt(p -> 1)

));

// Visitas por mes (últimos 6 meses)

Map<String, Integer> visitasPorMes = new LinkedHashMap<>();

SimpleDateFormat sdfMes = new SimpleDateFormat("MMM");

for (int i = 5; i >= 0; i--) {

cal.setTime(new Date());

cal.add(Calendar.MONTH, -i);

String mes = sdfMes.format(cal.getTime());

visitasPorMes.put(mes, 0);

}

proveedores.stream()

.filter(p -> p.getUltimaVisita() != null)

.forEach(p -> {

String mes = sdfMes.format(p.getUltimaVisita());

visitasPorMes.merge(mes, 1, Integer::sum);

});

// Actualizar la vista en el hilo de EDT

SwingUtilities.invokeLater(() -> {

try {

panelProveedores.actualizarTablaProveedores(proveedores);

panelProveedores.actualizarEstadisticas(total, visitadosEsteMes, ultimaVisita);

panelProveedores.actualizarGraficas(visitasPorMes, distribucionProductos);

} catch (Exception e) {

System.err.println("[UI] Error al actualizar componentes: " + e.getMessage());

e.printStackTrace();

}

});

}

public void filtrarProveedores(String filtro) {

System.out.println("[DEBUG] Filtrando proveedores por: " + filtro);

try {

List<Proveedor> proveedoresFiltrados;

switch (filtro) {

case "Con Visita Reciente":

Calendar cal = Calendar.getInstance();

cal.add(Calendar.MONTH, -1);

Date haceUnMes = new Date(cal.getTimeInMillis());

proveedoresFiltrados = proveedorDAO.obtenerTodosProveedores().stream()

.filter(p -> p.getUltimaVisita() != null &&

p.getUltimaVisita().after(haceUnMes))

.collect(Collectors.toList());

break;

case "Sin Visita Reciente":

cal = Calendar.getInstance();

cal.add(Calendar.MONTH, -1);

haceUnMes = new Date(cal.getTimeInMillis());

proveedoresFiltrados = proveedorDAO.obtenerTodosProveedores().stream()

.filter(p -> p.getUltimaVisita() == null ||

p.getUltimaVisita().before(haceUnMes))

.collect(Collectors.toList());

break;

case "Por Producto":

String producto = JOptionPane.showInputDialog("Ingrese el producto:");

if (producto == null || producto.isEmpty()) {

System.out.println("[DEBUG] Filtro por producto cancelado");

return;

}

proveedoresFiltrados = proveedorDAO.obtenerTodosProveedores().stream()

.filter(p -> p.getProductoSuministrado() != null &&

p.getProductoSuministrado().toLowerCase().contains(producto.toLowerCase()))

.collect(Collectors.toList());

break;

default: // "Todos"

proveedoresFiltrados = proveedorDAO.obtenerTodosProveedores();

break;

}

actualizarVistaProveedores(proveedoresFiltrados);

} catch (Exception e) {

mostrarErrorEnPanel("Error al filtrar proveedores: " + e.getMessage());

}

}

public void exportarReporteProveedores(String formato) {

try {

List<Proveedor> proveedores = proveedorDAO.obtenerTodosProveedores();

// Calcular estadísticas para el reporte

int totalProveedores = proveedores.size();

Calendar cal = Calendar.getInstance();

cal.set(Calendar.DAY\_OF\_MONTH, 1);

Date inicioMes = new Date(cal.getTimeInMillis());

int visitadosEsteMes = (int) proveedores.stream()

.filter(p -> p.getUltimaVisita() != null &&

p.getUltimaVisita().after(inicioMes))

.count();

// Distribución por producto suministrado

Map<String, Integer> distribucionProductos = proveedores.stream()

.collect(Collectors.groupingBy(

p -> p.getProductoSuministrado() != null && !p.getProductoSuministrado().isEmpty() ?

p.getProductoSuministrado() : "No especificado",

Collectors.summingInt(p -> 1)

));

// Visitas por mes (últimos 6 meses)

Map<String, Integer> visitasPorMes = new LinkedHashMap<>();

SimpleDateFormat sdfMes = new SimpleDateFormat("MMM");

for (int i = 5; i >= 0; i--) {

cal.setTime(new Date());

cal.add(Calendar.MONTH, -i);

String mes = sdfMes.format(cal.getTime());

visitasPorMes.put(mes, 0);

}

proveedores.stream()

.filter(p -> p.getUltimaVisita() != null)

.forEach(p -> {

String mes = sdfMes.format(p.getUltimaVisita());

visitasPorMes.merge(mes, 1, Integer::sum);

});

// Seleccionar el método de exportación según el formato

switch (formato.toUpperCase()) {

case "PDF":

exportarProveedoresAPDF(proveedores, totalProveedores, visitadosEsteMes,

visitasPorMes, distribucionProductos);

break;

case "EXCEL":

exportarProveedoresAExcel(proveedores, totalProveedores, visitadosEsteMes,

visitasPorMes, distribucionProductos);

break;

case "HTML":

exportarProveedoresAHTML(proveedores, totalProveedores, visitadosEsteMes,

visitasPorMes, distribucionProductos);

break;

default:

mostrarErrorEnPanel("Formato no soportado: " + formato);

}

} catch (Exception e) {

mostrarErrorEnPanel("Error al exportar reporte: " + e.getMessage());

e.printStackTrace();

}

}

private void exportarProveedoresAPDF(List<Proveedor> proveedores, int totalProveedores, int visitadosEsteMes,

Map<String, Integer> visitasPorMes,

Map<String, Integer> distribucionProductos) throws Exception {

String filename = "reporte\_proveedores\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".pdf";

Document document = new Document(PageSize.A4.rotate());

PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(filename));

document.open();

// Encabezado

agregarEncabezadoReporte(document, "PROVEEDORES", null, null);

// Resumen estadístico

PdfPTable summaryTable = new PdfPTable(3);

summaryTable.setWidthPercentage(100);

summaryTable.setSpacingBefore(10f);

addSummaryHeaderCell(summaryTable, "Total Proveedores");

addSummaryHeaderCell(summaryTable, "Visitados este mes");

addSummaryHeaderCell(summaryTable, "Por visitar este mes");

addSummaryDataCell(summaryTable, String.valueOf(totalProveedores));

addSummaryDataCell(summaryTable, String.valueOf(visitadosEsteMes));

addSummaryDataCell(summaryTable, String.valueOf(totalProveedores - visitadosEsteMes));

document.add(summaryTable);

// Tabla de detalle

Paragraph detalleTitle = new Paragraph("Detalle de Proveedores",

new Font(Font.FontFamily.HELVETICA, 12, Font.BOLD));

detalleTitle.setSpacingBefore(15f);

document.add(detalleTitle);

PdfPTable table = new PdfPTable(6);

table.setWidthPercentage(100);

table.setWidths(new float[]{1f, 2f, 1.5f, 2f, 2f, 2f});

// Encabezados

addDetailHeaderCell(table, "ID");

addDetailHeaderCell(table, "Nombre");

addDetailHeaderCell(table, "Teléfono");

addDetailHeaderCell(table, "Dirección");

addDetailHeaderCell(table, "Producto Suministrado");

addDetailHeaderCell(table, "Última Visita");

// Datos

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (Proveedor proveedor : proveedores) {

addDetailDataCell(table, proveedor.getId());

addDetailDataCell(table, proveedor.getNombre());

addDetailDataCell(table, proveedor.getTelefono());

addDetailDataCell(table, proveedor.getDireccion());

addDetailDataCell(table, proveedor.getProductoSuministrado());

addDetailDataCell(table, proveedor.getUltimaVisita() != null ?

sdf.format(proveedor.getUltimaVisita()) : "Nunca");

}

document.add(table);

// Gráficas

agregarGraficasProveedores(document, writer, visitasPorMes, distribucionProductos);

document.close();

// Abrir el archivo generado

abrirArchivoGenerado(filename);

}

private void agregarGraficasProveedores(Document document, PdfWriter writer,

Map<String, Integer> visitasPorMes,

Map<String, Integer> distribucionProductos)

throws IOException, DocumentException {

// 1. Gráfico de visitas por mes

DefaultCategoryDataset datasetVisitas = new DefaultCategoryDataset();

visitasPorMes.forEach((mes, cantidad) -> {

datasetVisitas.addValue(cantidad, "Visitas", mes);

});

JFreeChart chartVisitas = ChartFactory.createBarChart(

"Visitas por Mes",

"Mes",

"Número de Visitas",

datasetVisitas

);

// 2. Gráfico de distribución por producto

DefaultPieDataset datasetProductos = new DefaultPieDataset();

distribucionProductos.forEach(datasetProductos::setValue);

JFreeChart chartProductos = ChartFactory.createPieChart(

"Distribución por Producto",

datasetProductos,

true, true, false

);

// Configuración de tamaño y calidad

int width = 500;

int height = 300;

float quality = 1.0f;

// Agregar gráficos al documento

Paragraph chartTitle = new Paragraph("Análisis Gráfico",

new Font(Font.FontFamily.HELVETICA, 14, Font.BOLD));

chartTitle.setSpacingBefore(20f);

document.add(chartTitle);

// Gráfico de visitas

BufferedImage visitasImage = chartVisitas.createBufferedImage(width, height);

Image visitasPdfImage = Image.getInstance(writer, visitasImage, quality);

visitasPdfImage.setAlignment(Image.MIDDLE);

document.add(visitasPdfImage);

// Espacio entre gráficos

document.add(Chunk.NEWLINE);

// Gráfico de productos

BufferedImage productosImage = chartProductos.createBufferedImage(width, height);

Image productosPdfImage = Image.getInstance(writer, productosImage, quality);

productosPdfImage.setAlignment(Image.MIDDLE);

document.add(productosPdfImage);

}

private void exportarProveedoresAExcel(List<Proveedor> proveedores, int totalProveedores, int visitadosEsteMes,

Map<String, Integer> visitasPorMes,

Map<String, Integer> distribucionProductos) throws Exception {

String filename = "reporte\_proveedores\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".xlsx";

try (Workbook workbook = new XSSFWorkbook()) {

Sheet sheet = workbook.createSheet("Proveedores");

// Estilos

CellStyle headerStyle = crearEstiloEncabezado(workbook);

CellStyle titleStyle = crearEstiloTitulo(workbook);

CellStyle dateStyle = crearEstiloFecha(workbook);

int currentRow = 0;

// Título

Row titleRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, titleRow, 0, 5, "REPORTE DE PROVEEDORES", titleStyle);

// Fecha

Row dateRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, dateRow, 0, 5,

"Generado: " + new SimpleDateFormat("yyyy-MM-dd HH:mm").format(new Date()), null);

currentRow++;

// Resumen estadístico

Row statsTitleRow = sheet.createRow(currentRow++);

crearCeldaConMergedRegion(sheet, statsTitleRow, 0, 5, "RESUMEN ESTADÍSTICO", titleStyle);

Row statsHeaderRow = sheet.createRow(currentRow++);

crearCelda(statsHeaderRow, 0, "Total Proveedores", headerStyle);

crearCelda(statsHeaderRow, 1, "Visitados este mes", headerStyle);

crearCelda(statsHeaderRow, 2, "Por visitar este mes", headerStyle);

Row statsDataRow = sheet.createRow(currentRow++);

crearCelda(statsDataRow, 0, totalProveedores, null);

crearCelda(statsDataRow, 1, visitadosEsteMes, null);

crearCelda(statsDataRow, 2, totalProveedores - visitadosEsteMes, null);

currentRow++;

// Encabezados de la tabla

Row headerRow = sheet.createRow(currentRow++);

String[] headers = {"ID", "Nombre", "Teléfono", "Dirección", "Producto Suministrado", "Última Visita"};

for (int i = 0; i < headers.length; i++) {

crearCelda(headerRow, i, headers[i], headerStyle);

}

// Datos de proveedores

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (Proveedor proveedor : proveedores) {

Row row = sheet.createRow(currentRow++);

crearCelda(row, 0, proveedor.getId(), null);

crearCelda(row, 1, proveedor.getNombre(), null);

crearCelda(row, 2, proveedor.getTelefono(), null);

crearCelda(row, 3, proveedor.getDireccion(), null);

crearCelda(row, 4, proveedor.getProductoSuministrado(), null);

if (proveedor.getUltimaVisita() != null) {

Cell fechaCell = row.createCell(5);

fechaCell.setCellValue(proveedor.getUltimaVisita());

fechaCell.setCellStyle(dateStyle);

} else {

crearCelda(row, 5, "Nunca", null);

}

}

// Autoajustar columnas

for (int i = 0; i < headers.length; i++) {

sheet.autoSizeColumn(i);

}

// Hoja adicional para gráficos

Sheet chartSheet = workbook.createSheet("Gráficos");

currentRow = 0;

// Datos para gráfico de visitas por mes

Row chartTitleRow = chartSheet.createRow(currentRow++);

crearCeldaConMergedRegion(chartSheet, chartTitleRow, 0, 1, "Visitas por Mes", titleStyle);

Row chartHeaderRow = chartSheet.createRow(currentRow++);

crearCelda(chartHeaderRow, 0, "Mes", headerStyle);

crearCelda(chartHeaderRow, 1, "Visitas", headerStyle);

for (Map.Entry<String, Integer> entry : visitasPorMes.entrySet()) {

Row dataRow = chartSheet.createRow(currentRow++);

crearCelda(dataRow, 0, entry.getKey(), null);

crearCelda(dataRow, 1, entry.getValue(), null);

}

// Datos para gráfico de distribución por producto

currentRow += 2;

Row chartTitleRow2 = chartSheet.createRow(currentRow++);

crearCeldaConMergedRegion(chartSheet, chartTitleRow2, 0, 1, "Distribución por Producto", titleStyle);

Row chartHeaderRow2 = chartSheet.createRow(currentRow++);

crearCelda(chartHeaderRow2, 0, "Producto", headerStyle);

crearCelda(chartHeaderRow2, 1, "Cantidad", headerStyle);

for (Map.Entry<String, Integer> entry : distribucionProductos.entrySet()) {

Row dataRow = chartSheet.createRow(currentRow++);

crearCelda(dataRow, 0, entry.getKey(), null);

crearCelda(dataRow, 1, entry.getValue(), null);

}

// Guardar archivo

try (FileOutputStream outputStream = new FileOutputStream(filename)) {

workbook.write(outputStream);

}

}

abrirArchivoGenerado(filename);

}

private void exportarProveedoresAHTML(List<Proveedor> proveedores, int totalProveedores, int visitadosEsteMes,

Map<String, Integer> visitasPorMes,

Map<String, Integer> distribucionProductos) throws Exception {

String filename = "reporte\_proveedores\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".html";

try (PrintWriter writer = new PrintWriter(new FileOutputStream(filename))) {

writer.println("<!DOCTYPE html>");

writer.println("<html lang='es'>");

writer.println("<head>");

writer.println("<meta charset='UTF-8'>");

writer.println("<title>Reporte de Proveedores</title>");

writer.println("<style>");

writer.println("body { font-family: Arial, sans-serif; margin: 20px; }");

writer.println("h1 { color: #2c3e50; }");

writer.println("table { width: 100%; border-collapse: collapse; margin-bottom: 20px; }");

writer.println("th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }");

writer.println("th { background-color: #f2f2f2; }");

writer.println(".resumen { background-color: #f8f9fa; padding: 15px; margin-bottom: 20px; }");

writer.println(".chart-container { display: flex; justify-content: space-around; margin: 20px 0; }");

writer.println(".chart { width: 45%; border: 1px solid #ddd; padding: 10px; }");

writer.println("</style>");

writer.println("</head>");

writer.println("<body>");

writer.println("<h1>Reporte de Proveedores</h1>");

writer.println("<div class='resumen'>");

writer.println("<p><strong>Generado:</strong> " + new Date() + "</p>");

writer.println("<p><strong>Total Proveedores:</strong> " + totalProveedores + "</p>");

writer.println("<p><strong>Visitados este mes:</strong> " + visitadosEsteMes + "</p>");

writer.println("<p><strong>Por visitar este mes:</strong> " + (totalProveedores - visitadosEsteMes) + "</p>");

writer.println("</div>");

writer.println("<h2>Detalle de Proveedores</h2>");

writer.println("<table>");

writer.println("<tr><th>ID</th><th>Nombre</th><th>Teléfono</th><th>Dirección</th><th>Producto Suministrado</th><th>Última Visita</th></tr>");

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (Proveedor proveedor : proveedores) {

writer.println(String.format(

"<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>",

proveedor.getId(),

proveedor.getNombre(),

proveedor.getTelefono(),

proveedor.getDireccion(),

proveedor.getProductoSuministrado(),

proveedor.getUltimaVisita() != null ? sdf.format(proveedor.getUltimaVisita()) : "Nunca"

));

}

writer.println("</table>");

// Sección de gráficos

writer.println("<div class='chart-container'>");

writer.println("<div class='chart'>");

writer.println("<h3>Visitas por Mes</h3>");

writer.println("<ul>");

visitasPorMes.forEach((mes, cantidad) -> {

writer.println(String.format("<li><strong>%s:</strong> %d visitas</li>", mes, cantidad));

});

writer.println("</ul>");

writer.println("</div>");

writer.println("<div class='chart'>");

writer.println("<h3>Distribución por Producto</h3>");

writer.println("<ul>");

distribucionProductos.forEach((producto, cantidad) -> {

writer.println(String.format("<li><strong>%s:</strong> %d proveedores</li>", producto, cantidad));

});

writer.println("</ul>");

writer.println("</div>");

writer.println("</div>");

writer.println("</body>");

writer.println("</html>");

}

abrirArchivoGenerado(filename);

}

public void reimprimirReporte(String tipoReporte, String idReporte) {

try {

File reporte = buscarArchivoReporte(tipoReporte, idReporte);

if (reporte == null) {

// Intenta regenerar el reporte si no se encuentra

regenerarReporte(tipoReporte, idReporte);

reporte = buscarArchivoReporte(tipoReporte, idReporte);

if (reporte == null) {

mostrarErrorEnPanel("No se encontró el reporte " + idReporte +

" de tipo " + tipoReporte);

return;

}

}

// Abrir el archivo

if (Desktop.isDesktopSupported()) {

Desktop.getDesktop().open(reporte);

} else {

mostrarErrorEnPanel("No se puede abrir el reporte automáticamente");

}

} catch (Exception e) {

mostrarErrorEnPanel("Error al reimprimir: " + e.getMessage());

e.printStackTrace();

}

}

private void regenerarReporte(String tipoReporte, String idReporte) {

switch (tipoReporte.toUpperCase()) {

case "VENTA":

try {

reimprimirTicketVenta(idReporte);

} catch (Exception e) {

}

break;

case "TICKET":

try {

reimprimirTicketVenta(idReporte);

} catch (Exception e) {

mostrarErrorEnPanel("Error al regenerar el ticket: " + e.getMessage());

}

break;

case "INVENTARIO":

exportarReporteInventario("PDF");

break;

case "CLIENTES":

exportarReporteClientes("PDF");

break;

case "PROVEEDORES":

exportarReporteProveedores("PDF");

break;

default:

mostrarErrorEnPanel("Tipo de reporte no soportado: " + tipoReporte);

}

}

public void reimprimirTicketVenta(String idVenta) throws Exception {

// Obtener los datos de la venta desde la BD

Venta venta = obtenerVentaPorId(idVenta);

if (venta == null) {

throw new Exception("No se encontró la venta con ID: " + idVenta);

}

// Generar el ticket nuevamente

VentaContro ventaControl = new VentaContro(usuario);

ventaControl.generarTicketVenta(venta);

}

private Venta obtenerVentaPorId(String idVenta) throws SQLException {

Connection conn = null;

PreparedStatement pstmtVenta = null;

PreparedStatement pstmtDetalles = null;

ResultSet rsVenta = null;

ResultSet rsDetalles = null;

try {

conn = ConexionAccess.conectar();

// Consulta para obtener la venta principal

String sqlVenta = "SELECT id, fecha, total, metodo\_pago, descuento, monto\_recibido FROM Ventas WHERE id = ?";

pstmtVenta = conn.prepareStatement(sqlVenta);

pstmtVenta.setString(1, idVenta);

rsVenta = pstmtVenta.executeQuery();

if (!rsVenta.next()) {

return null;

}

Venta venta = new Venta();

venta.setId(rsVenta.getInt("id"));

venta.setFecha(rsVenta.getTimestamp("fecha"));

venta.setTotal(rsVenta.getDouble("total"));

venta.setMetodoPago(rsVenta.getString("metodo\_pago"));

venta.setDescuento(rsVenta.getDouble("descuento"));

venta.setMontoRecibido(rsVenta.getDouble("monto\_recibido"));

// Obtener detalles de la venta

String sqlDetalles = "SELECT p.id, p.nombre, p.precio\_venta, dv.cantidad " +

"FROM DetalleVenta dv " +

"JOIN Productos p ON dv.id\_producto = p.id " +

"WHERE dv.id\_venta = ?";

pstmtDetalles = conn.prepareStatement(sqlDetalles);

pstmtDetalles.setString(1, idVenta);

rsDetalles = pstmtDetalles.executeQuery();

List<Producto> productos = new ArrayList<>();

while (rsDetalles.next()) {

Producto producto = new Producto(

rsDetalles.getString("id"),

rsDetalles.getString("nombre"),

"", "", "",

rsDetalles.getInt("cantidad"),

0, 0, 0.0,

rsDetalles.getDouble("precio\_venta"),

false, 0.0, null, "", "", ""

);

productos.add(producto);

}

venta.setProductos(productos);

return venta;

} finally {

// Cerrar recursos

if (rsDetalles != null) rsDetalles.close();

if (rsVenta != null) rsVenta.close();

if (pstmtDetalles != null) pstmtDetalles.close();

if (pstmtVenta != null) pstmtVenta.close();

if (conn != null) conn.close();

}

}

private void reimprimirReporteGenerado(String tipoReporte, String idReporte) throws Exception {

// Buscar el archivo del reporte en la estructura de carpetas

File reporte = buscarArchivoReporte(tipoReporte, idReporte);

if (reporte == null || !reporte.exists()) {

throw new Exception("No se encontró el archivo del reporte");

}

// Abrir el archivo con el visor PDF predeterminado

if (Desktop.isDesktopSupported()) {

Desktop.getDesktop().open(reporte);

} else {

throw new Exception("No se puede abrir el reporte automáticamente en este sistema");

}

}

public boolean existeReporte(String tipoReporte, String idReporte) {

return buscarArchivoReporte(tipoReporte, idReporte) != null;

}

private File buscarArchivoReporte(String tipoReporte, String idReporte) {

// Ruta base donde se guardan los reportes

String basePath = "C:\\Users\\Anahi\\eclipse-workspace\\punto\_venta\_2\\Reportes";

// Primero buscar en la subcarpeta específica del tipo de reporte

File carpetaTipo = new File(basePath, tipoReporte);

if (carpetaTipo.exists() && carpetaTipo.isDirectory()) {

File[] archivos = buscarArchivosEnCarpeta(carpetaTipo, idReporte);

if (archivos != null && archivos.length > 0) {

return archivos[0]; // Devolver el más reciente

}

}

// Si no se encontró en la subcarpeta, buscar en la carpeta principal de Reportes

File carpetaPrincipal = new File(basePath);

if (carpetaPrincipal.exists() && carpetaPrincipal.isDirectory()) {

File[] archivos = buscarArchivosEnCarpeta(carpetaPrincipal, idReporte);

if (archivos != null && archivos.length > 0) {

return archivos[0];

}

}

return null;

}

private File[] buscarArchivosEnCarpeta(File carpeta, String idReporte) {

// Patrones de nombres de archivo a buscar

String[] patrones = {

"reporte\_" + tipoReporteActual.toLowerCase() + "\_" + idReporte + ".\*",

tipoReporteActual.toLowerCase() + "\_" + idReporte + ".\*",

idReporte + ".\*"

};

// Filtrar archivos que coincidan con los patrones

File[] archivos = carpeta.listFiles((dir, name) -> {

for (String patron : patrones) {

if (name.toLowerCase().matches(patron.toLowerCase().replace(".", "\\.").replace("\*", ".\*"))) {

return true;

}

}

return false;

});

// Ordenar por fecha de modificación (más reciente primero)

if (archivos != null && archivos.length > 0) {

Arrays.sort(archivos, (f1, f2) -> Long.compare(f2.lastModified(), f1.lastModified()));

// Verificar integridad de los archivos

for (File archivo : archivos) {

if (verificarIntegridadArchivo(archivo)) {

return new File[]{archivo}; // Devolver solo el primero válido

}

}

}

return null;

}

private boolean verificarIntegridadArchivo(File archivo) {

try {

// Verificaciones básicas

if (!archivo.exists() || archivo.length() == 0) {

return false;

}

// Verificación específica por tipo de archivo

String nombre = archivo.getName().toLowerCase();

if (nombre.endsWith(".pdf")) {

// Verificar si es un PDF válido

try (RandomAccessFile raf = new RandomAccessFile(archivo, "r")) {

byte[] buffer = new byte[4];

raf.read(buffer);

return new String(buffer).equals("%PDF");

}

} else if (nombre.endsWith(".xlsx")) {

// Verificación básica para Excel

return archivo.length() > 100; // Tamaño mínimo razonable

} else if (nombre.endsWith(".html")) {

// Verificación básica para HTML

try (BufferedReader reader = new BufferedReader(new FileReader(archivo))) {

String primeraLinea = reader.readLine();

return primeraLinea != null && primeraLinea.trim().toLowerCase().startsWith("<!doctype html");

}

}

return true; // Para otros tipos de archivo, asumir que son válidos

} catch (Exception e) {

return false;

}

}

private boolean coincideConReporte(String nombreArchivo, String tipoReporte, String idReporte) {

String nombreLower = nombreArchivo.toLowerCase();

String tipoLower = tipoReporte.toLowerCase();

String idLower = idReporte.toLowerCase();

// Patrones que puede tener un nombre de reporte

boolean patron1 = nombreLower.contains(tipoLower) && nombreLower.contains(idLower);

boolean patron2 = nombreLower.startsWith(tipoLower) && nombreLower.contains(idLower);

boolean patron3 = nombreLower.endsWith(idLower + ".pdf");

// Formatos aceptados

boolean formatoValido = nombreLower.endsWith(".pdf") ||

nombreLower.endsWith(".html") ||

nombreLower.endsWith(".xlsx");

return (patron1 || patron2 || patron3) && formatoValido;

}

private void reimprimirReporteInventario(String idReporte) throws Exception {

// Buscar el archivo del reporte en la estructura de carpetas

File reporte = buscarArchivoReporte("INVENTARIO", idReporte);

if (reporte == null || !reporte.exists()) {

// Si no existe el archivo, generamos un nuevo reporte PDF con los datos actuales

String filename = "reporte\_inventario\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".pdf";

exportarInventarioAPDF(

inventarioDAO.obtenerTodosProductos(),

inventarioDAO.obtenerTodosProductos().size(),

inventarioDAO.buscarProductosNecesitanReposicion().size(),

inventarioDAO.buscarProductosSinStock().size(),

calcularStockPromedio(),

obtenerDistribucionCategorias()

);

reporte = new File(filename);

}

// Abrir el archivo con el visor predeterminado

if (Desktop.isDesktopSupported()) {

Desktop.getDesktop().open(reporte);

} else {

throw new Exception("No se puede abrir el reporte automáticamente en este sistema");

}

}

private double calcularStockPromedio() {

List<Producto> productos = inventarioDAO.obtenerTodosProductos();

return productos.stream()

.mapToInt(Producto::getCantidadDisponible)

.average()

.orElse(0);

}

private Map<String, Integer> obtenerDistribucionCategorias() {

return inventarioDAO.obtenerTodosProductos().stream()

.collect(Collectors.groupingBy(

Producto::getCategoria,

Collectors.summingInt(p -> 1)

));

}

private void reimprimirReporteClientes(String idReporte) throws Exception {

// Buscar el archivo del reporte en la estructura de carpetas

File reporte = buscarArchivoReporte("CLIENTES", idReporte);

if (reporte == null || !reporte.exists()) {

// Si no existe el archivo, generamos un nuevo reporte PDF con los datos actuales

String filename = "reporte\_clientes\_" + new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date()) + ".pdf";

List<Cliente> clientes = clienteDAO.obtenerTodos();

int totalClientes = clientes.size();

int activos = (int) clientes.stream().filter(c -> c.getFechaEliminacion() == null).count();

Map<String, Integer> registrosPorMes = clientes.stream()

.collect(Collectors.groupingBy(

c -> new SimpleDateFormat("MMM").format(c.getFechaRegistro()),

Collectors.summingInt(c -> 1)

));

exportarClientesAPDF(clientes, totalClientes, activos, registrosPorMes);

reporte = new File(filename);

}

// Abrir el archivo con el visor predeterminado

if (Desktop.isDesktopSupported()) {

Desktop.getDesktop().open(reporte);

} else {

throw new Exception("No se puede abrir el reporte automáticamente en este sistema");

}

}

}

package Vista;

import Modelo.Usuario;

import Modelo.Venta;

import Modelo.ClienteDAO;

import Modelo.ClienteDAOImpl;

import Modelo.Producto;

import Controlador.ReportesControlador;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.net.URL;

import java.text.SimpleDateFormat;

import javax.swing.border.\*;

import com.toedter.calendar.JDateChooser;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.List;

import java.util.Objects;

public class reportes extends JFrame {

private Usuario usuario;

private String rolUsuario;

private JPanel panelContenido; // Panel dinámico para cambiar vistas

private CardLayout cardLayout;

private final ReportesControlador controlador;

private ReporteVentasPanel panelVentas;

private ReporteInventarioPanel panelInventario;

private ReporteClientePanel panelClientes;

// Constructor

public reportes(Usuario usuario, ReportesControlador controlador) {

this.usuario = usuario;

this.controlador = controlador;

// Primero crear la interfaz

initUI();

// Luego crear los paneles

this.panelVentas = new ReporteVentasPanel(usuario, controlador);

this.panelInventario = new ReporteInventarioPanel(usuario, controlador, cardLayout, panelContenido);

this.panelClientes = new ReporteClientePanel(usuario, controlador, cardLayout, panelContenido);

// Registrar paneles

panelContenido.add(panelVentas, "reporte\_ventas");

panelContenido.add(panelInventario, "reporte\_inventario");

panelContenido.add(panelClientes, "reporte\_clientes");

// Configurar paneles en el controlador

controlador.setPanelVentas(panelVentas);

controlador.setPanelInventario(panelInventario);

controlador.setPanelClientes(panelClientes);

}

private void initUI() {

setTitle("El Habanerito - Reportes");

setSize(1517, 903);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

setResizable(true);

JPanel mainPanel = new JPanel(new BorderLayout());

// Panel superior con encabezado y menú

JPanel topContainer = new JPanel();

topContainer.setLayout(new BoxLayout(topContainer, BoxLayout.Y\_AXIS));

topContainer.add(createHeaderPanel());

topContainer.add(crearMenuHorizontal());

mainPanel.add(topContainer, BorderLayout.NORTH);

// Panel central dinámico

panelContenido = new JPanel();

cardLayout = new CardLayout();

panelContenido.setLayout(cardLayout);

// Agregar vistas

panelContenido.add(crearMenuPrincipalPanel(), "menu\_principal");

mainPanel.add(panelContenido, BorderLayout.CENTER);

add(mainPanel);

}

private JPanel crearMenuPrincipalPanel() {

JPanel menuPanel = new JPanel(new GridBagLayout());

menuPanel.setBorder(BorderFactory.createEmptyBorder(30, 30, 30, 30));

menuPanel.setBackground(new Color(240, 240, 240));

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(20, 20, 20, 20);

gbc.fill = GridBagConstraints.BOTH;

gbc.weightx = 1;

gbc.weighty = 1;

// Tarjeta de Reporte de Ventas (existente)

gbc.gridx = 0;

gbc.gridy = 0;

JPanel cardVentas = crearTarjetaReporte(

"VENTAS",

"/imagen/ventas\_icon.png",

"Reportes detallados de ventas por período",

new Color(144, 238, 144)

);

cardVentas.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

mostrarReporteVentas();

}

});

menuPanel.add(cardVentas, gbc);

// Tarjeta de Reporte de Inventario (existente)

gbc.gridx = 1;

JPanel cardInventario = crearTarjetaReporte(

"INVENTARIO",

"/imagen/inventario\_icon.png",

"Estado actual del inventario y alertas",

new Color(135, 206, 250)

);

cardInventario.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

mostrarReporteInventario();

}

});

menuPanel.add(cardInventario, gbc);

// Tarjeta de Reporte de Clientes (existente)

gbc.gridx = 0;

gbc.gridy = 1;

JPanel cardClientes = crearTarjetaReporte(

"CLIENTES",

"/imagen/clientes\_icon.png",

"Comportamiento y fidelidad de clientes",

new Color(255, 182, 193)

);

cardClientes.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

mostrarReporteClientes();

}

});

menuPanel.add(cardClientes, gbc);

// Tarjeta de Reporte de Proveedores (existente)

gbc.gridx = 1;

JPanel cardProveedores = crearTarjetaReporte(

"PROVEEDORES",

"/imagen/proveedores\_icon.png",

"Desempeño y relación con proveedores",

new Color(221, 160, 221)

);

cardProveedores.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

mostrarReporteProveedores();

}

});

menuPanel.add(cardProveedores, gbc);

// --- NUEVA TARJETA PARA REIMPRESIÓN ---

gbc.gridx = 0;

gbc.gridy = 2;

gbc.gridwidth = 2; // Ocupa 2 columnas

JPanel cardReimpresion = crearTarjetaReporte(

"REIMPRESIÓN",

"/imagen/printer\_icon.png",

"Reimprimir tickets de venta o reportes",

new Color(255, 215, 0) // Color dorado para distinguirlo

);

cardReimpresion.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

new ReimprimirDialog(usuario,reportes.this, controlador).setVisible(true);

}

});

menuPanel.add(cardReimpresion, gbc);

return menuPanel;

}

private JPanel crearTarjetaReporte(String titulo, String icono, String descripcion, Color color) {

JPanel card = new JPanel(new BorderLayout());

card.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(color.darker(), 2),

BorderFactory.createEmptyBorder(25, 25, 25, 25)

));

card.setBackground(color);

card.setPreferredSize(new Dimension(350, 250));

// Efecto hover

card.addMouseListener(new MouseAdapter() {

public void mouseEntered(MouseEvent e) {

card.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(Color.BLACK, 2),

BorderFactory.createEmptyBorder(25, 25, 25, 25)

));

card.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

}

public void mouseExited(MouseEvent e) {

card.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(color.darker(), 2),

BorderFactory.createEmptyBorder(25, 25, 25, 25)

));

card.setCursor(Cursor.getDefaultCursor());

}

});

JLabel lblTitulo = new JLabel(titulo, SwingConstants.CENTER);

lblTitulo.setFont(new Font("Arial", Font.BOLD, 24));

lblTitulo.setForeground(Color.DARK\_GRAY);

JLabel lblDesc = new JLabel("<html><div style='text-align: center;'>" + descripcion + "</div></html>", SwingConstants.CENTER);

lblDesc.setFont(new Font("Arial", Font.PLAIN, 14));

lblDesc.setForeground(Color.DARK\_GRAY);

// Intenta cargar el icono

try {

URL imgUrl = getClass().getResource(icono);

if (imgUrl != null) {

ImageIcon originalIcon = new ImageIcon(imgUrl);

Image scaledImage = originalIcon.getImage().getScaledInstance(80, 80, Image.SCALE\_SMOOTH);

JLabel lblIcono = new JLabel(new ImageIcon(scaledImage), SwingConstants.CENTER);

card.add(lblIcono, BorderLayout.NORTH);

}

} catch (Exception e) {

System.out.println("No se pudo cargar el icono: " + icono);

card.add(Box.createVerticalStrut(30), BorderLayout.NORTH);

}

card.add(lblTitulo, BorderLayout.CENTER);

card.add(lblDesc, BorderLayout.SOUTH);

return card;

}

public void mostrarMenuPrincipalReportes() {

cardLayout.show(panelContenido, "menu\_principal");

}

private void mostrarReporteVentas() {

try {

System.out.println("[DEBUG] Intentando mostrar reporte de ventas");

// Verificar si el panel ya existe

Component ventasPanel = findPanel("reporte\_ventas");

if (ventasPanel == null) {

System.out.println("[DEBUG] Creando nuevo panel de ventas");

ventasPanel = new ReporteVentasPanel(usuario, controlador);

// Crear panel con botón de regreso

JPanel panelConBoton = new JPanel(new BorderLayout());

panelConBoton.add(ventasPanel, BorderLayout.CENTER);

JButton btnRegresar = new JButton("Regresar al Menú");

btnRegresar.addActionListener(e -> mostrarMenuPrincipalReportes());

panelConBoton.add(btnRegresar, BorderLayout.SOUTH);

panelContenido.add(panelConBoton, "reporte\_ventas");

}

// Mostrar el panel

cardLayout.show(panelContenido, "reporte\_ventas");

System.out.println("[DEBUG] Panel de ventas mostrado");

// Actualizar la interfaz

revalidate();

repaint();

} catch (Exception e) {

System.err.println("[ERROR] Al mostrar ventas: " + e.getMessage());

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Error al abrir reporte de ventas", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void mostrarReporteInventario() {

Component inventarioPanel = findPanel("reporte\_inventario");

if (inventarioPanel == null) {

inventarioPanel = new ReporteInventarioPanel(

usuario,

controlador,

cardLayout,

panelContenido

);

panelContenido.add(inventarioPanel, "reporte\_inventario");

}

cardLayout.show(panelContenido, "reporte\_inventario");

}

private void mostrarReporteClientes() {

Component clientesPanel = findPanel("reporte\_clientes");

if (clientesPanel == null) {

clientesPanel = new ReporteClientePanel(

usuario,

controlador,

cardLayout,

panelContenido

);

panelContenido.add(clientesPanel, "reporte\_clientes");

// Configurar el controlador con el panel de clientes

controlador.setPanelClientes((ReporteClientePanel) clientesPanel);

// Cargar datos iniciales

controlador.cargarDatosClientes();

}

cardLayout.show(panelContenido, "reporte\_clientes");

}

private void mostrarReporteProveedores() {

Component proveedoresPanel = findPanel("reporte\_proveedores");

if (proveedoresPanel == null) {

proveedoresPanel = new ReporteProveedoresPanel(usuario, controlador, cardLayout, panelContenido);

panelContenido.add(proveedoresPanel, "reporte\_proveedores");

}

cardLayout.show(panelContenido, "reporte\_proveedores");

}

private Component findPanel(String name) {

for (Component comp : panelContenido.getComponents()) {

if (comp.getName() != null && comp.getName().equals(name)) {

return comp;

}

}

return null;

}

private JPanel createHeaderPanel() {

JPanel panel = new JPanel(new BorderLayout());

panel.setBackground(new Color(255, 198, 144));

panel.setPreferredSize(new Dimension(getWidth(), 80));

panel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 0));

rightPanel.setOpaque(false);

JButton usuarioBtn = new JButton(usuario.getUsername());

usuarioBtn.setFont(new Font("Arial", Font.BOLD, 14));

usuarioBtn.setForeground(Color.BLACK);

usuarioBtn.setContentAreaFilled(false);

usuarioBtn.setBorderPainted(false);

usuarioBtn.setFocusPainted(false);

usuarioBtn.setCursor(new Cursor(Cursor.HAND\_CURSOR));

usuarioBtn.addActionListener(e -> cambiarUsuario());

rightPanel.add(usuarioBtn);

try {

ImageIcon originalIcon = new ImageIcon("imagen\\logo.png");

Image originalImage = originalIcon.getImage();

int logoHeight = 60;

int logoWidth = (int) ((double) originalIcon.getIconWidth() / originalIcon.getIconHeight() \* logoHeight);

Image resizedImage = originalImage.getScaledInstance(logoWidth, logoHeight, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage));

rightPanel.add(logo, 0);

} catch (Exception e) {

System.err.println("Error cargando el logo: " + e.getMessage());

}

panel.add(rightPanel, BorderLayout.EAST);

return panel;

}

private JPanel crearMenuHorizontal() {

JPanel menuPanel = new JPanel(new GridLayout(1, 7));

menuPanel.setBackground(new Color(230, 230, 230));

menuPanel.setPreferredSize(new Dimension(0, 50));

menuPanel.setBorder(BorderFactory.createMatteBorder(1, 0, 1, 0, Color.GRAY));

String[] opciones = {"Productos", "Reportes", "Inventario", "Cliente", "Proveedores", "Usuarios", "Salir"};

for (String opcion : opciones) {

JButton btn = crearBotonMenu(opcion, opcion.equals("Reportes"));

btn.addActionListener(e -> manejarAccionMenu(opcion));

menuPanel.add(btn);

}

return menuPanel;

}

private JButton crearBotonMenu(String texto, boolean esActivo) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

boton.setPreferredSize(new Dimension(0, 50));

boton.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 50));

if (esActivo) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.BLACK);

} else {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

if (!esActivo) {

boton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseEntered(java.awt.event.MouseEvent evt) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

public void mouseExited(java.awt.event.MouseEvent evt) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

});

}

return boton;

}

private void manejarAccionMenu(String opcion) {

switch (opcion) {

case "Salir":

this.dispose();

new menuprincipal(usuario).setVisible(true);

break;

case "Productos":

this.dispose();

new producto(usuario).setVisible(true);

break;

case "Reportes":

// Ya estamos en reportes

break;

case "Inventario":

this.dispose();

new inventario(usuario).setVisible(true);

break;

case "Cliente":

this.dispose();

ClienteDAO clienteDAO = new ClienteDAOImpl();

new clientes(usuario, clienteDAO).setVisible(true);

break;

case "Proveedores":

this.dispose();

new proveedores(usuario).setVisible(true);

break;

case "Usuarios":

this.dispose();

new gestionUsuario(usuario).setVisible(true);

break;

}

}

private void cambiarUsuario() {

JDialog changeUserDialog = new JDialog(this, "Cambiar Usuario", true);

changeUserDialog.setSize(300, 150);

changeUserDialog.setLocationRelativeTo(this);

JPanel dialogPanel = new JPanel(new BorderLayout(10, 10));

dialogPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

JLabel instructionLabel = new JLabel("¿Desea cambiar de usuario?", SwingConstants.CENTER);

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 0));

JButton cambiarBtn = new JButton("Cambiar de Usuario");

JButton cancelarBtn = new JButton("Cancelar");

cambiarBtn.addActionListener(e -> {

Window[] windows = Window.getWindows();

for (Window window : windows) {

if (window instanceof producto) {

window.setVisible(false);

window.dispose();

}

}

new Login().setVisible(true);

changeUserDialog.dispose();

});

cancelarBtn.addActionListener(e -> changeUserDialog.dispose());

buttonPanel.add(cambiarBtn);

buttonPanel.add(cancelarBtn);

dialogPanel.add(instructionLabel, BorderLayout.CENTER);

dialogPanel.add(buttonPanel, BorderLayout.SOUTH);

changeUserDialog.getContentPane().add(dialogPanel);

changeUserDialog.setVisible(true);

}

public void mostrarError(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Error", JOptionPane.ERROR\_MESSAGE);

}

public void mostrarMensaje(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Información", JOptionPane.INFORMATION\_MESSAGE);

}

}

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

package Controlador;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.sql.SQLException;

import java.util.Date;

import java.util.List;

import com.itextpdf.text.\*;

import com.itextpdf.text.pdf.\*;

import Modelo.Venta;

public class PDFExporter implements PdfPageEvent {

private Font footerFont;

private ReportesControlador controlador;

private static final String REPORTS\_BASE\_DIR = "Reportes";

public PDFExporter(ReportesControlador controlador) {

this.controlador = controlador;

this.footerFont = new Font(Font.FontFamily.HELVETICA, 8, Font.NORMAL);

}

// Método principal para exportar

public void exportarAPDF(String filename, String tipoReporte,

Date fechaInicio, Date fechaFin) throws DocumentException, IOException, SQLException {

File file = new File(filename);

Document document = new Document(PageSize.A4.rotate());

PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(file));

// Crear el directorio de reportes si no existe

writer.setPageEvent(this);

document.open();

controlador.agregarEncabezadoReporte(document, tipoReporte, fechaInicio, fechaFin);

if ("VENTAS".equals(tipoReporte)) {

List<Venta> ventas = controlador.obtenerVentasDesdeBD(fechaInicio, fechaFin);

controlador.agregarReporteVentas(document, ventas, writer);

controlador.agregarGraficos(document, ventas, writer);

} else {

document.add(new Paragraph("Tipo de reporte no soportado para exportación"));

}

document.close();

if (!file.exists() || file.length() == 0) {

throw new IOException("No se pudo crear el archivo PDF");

}

}

// Implementación de los métodos de PdfPageEvent

@Override

public void onOpenDocument(PdfWriter writer, Document document) {

// No es necesario implementar

}

@Override

public void onStartPage(PdfWriter writer, Document document) {

// No es necesario implementar

}

@Override

public void onEndPage(PdfWriter writer, Document document) {

try {

agregarNumeroPagina(writer, document);

} catch (DocumentException e) {

System.err.println("Error al agregar número de página: " + e.getMessage());

}

}

@Override

public void onCloseDocument(PdfWriter writer, Document document) {

// No es necesario implementar

}

@Override

public void onParagraph(PdfWriter writer, Document document, float paragraphPosition) {

// No es necesario implementar

}

@Override

public void onParagraphEnd(PdfWriter writer, Document document, float paragraphPosition) {

// No es necesario implementar

}

@Override

public void onChapter(PdfWriter writer, Document document, float paragraphPosition, Paragraph title) {

// No es necesario implementar

}

@Override

public void onChapterEnd(PdfWriter writer, Document document, float paragraphPosition) {

// No es necesario implementar

}

@Override

public void onSection(PdfWriter writer, Document document, float paragraphPosition, int depth, Paragraph title) {

// No es necesario implementar

}

@Override

public void onSectionEnd(PdfWriter writer, Document document, float paragraphPosition) {

// No es necesario implementar

}

@Override

public void onGenericTag(PdfWriter writer, Document document, Rectangle rect, String text) {

// No es necesario implementar

}

private void agregarNumeroPagina(PdfWriter writer, Document document) throws DocumentException {

PdfContentByte cb = writer.getDirectContent();

Phrase footer = new Phrase(

String.format("Página %d", writer.getPageNumber()),

footerFont

);

ColumnText.showTextAligned(

cb,

Element.ALIGN\_CENTER,

footer,

(document.right() - document.left()) / 2 + document.leftMargin(),

document.bottom() - 15,

0

);

}

}

package Vista;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.EventQueue;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.GradientPaint;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.GridLayout;

import java.awt.RenderingHints;

import java.text.NumberFormat;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;

import javax.swing.BorderFactory;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTabbedPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.SwingConstants;

import javax.swing.Timer;

import javax.swing.border.EmptyBorder;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.chart.axis.CategoryAxis;

import org.jfree.chart.axis.CategoryLabelPositions;

import org.jfree.chart.axis.NumberAxis;

import org.jfree.chart.plot.CategoryPlot;

import org.jfree.chart.plot.PlotOrientation;

import org.jfree.chart.renderer.category.BarRenderer;

import org.jfree.chart.title.TextTitle;

import org.jfree.data.category.DefaultCategoryDataset;

import com.toedter.calendar.JDateChooser;

import Controlador.ReportesControlador;

import Modelo.Producto;

import Modelo.Usuario;

import Modelo.Venta;

import Vista.reportes;

public class ReporteVentasPanel extends JPanel {

private Usuario usuario;

private ReportesControlador controlador;

// Componentes para filtros

private JComboBox<String> comboFiltro;

private JDateChooser dateChooserInicio;

private JDateChooser dateChooserFin;

private JButton btnFiltrar;

// Componentes para exportación

private JComboBox<String> comboExportar;

private JButton btnExportar;

private JButton btnImprimir;

// Componentes de datos

private JTextArea txtAreaVentas;

private JTable tablaResumen;

private StatCard[] statsCards; // Para el gráfico principal de resumen

private ChartPanel graficoMetodosPanel; // Para el gráfico de métodos de pago

private ChartPanel graficoProductosPanel;

private ChartPanel graficoResumenPanel;

private ChartPanel chartPanel;

public ReporteVentasPanel(Usuario usuario, ReportesControlador controlador) {

this.controlador = controlador;

this.controlador.setPanelVentas(this); // ✅ ASÍ EVITAS EL NULL

initUI();

setControlador(controlador);

}

private void initUI() {

setLayout(new BorderLayout(10, 10));

setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

JPanel panelControles = crearPanelControles();

add(panelControles, BorderLayout.NORTH);

JTabbedPane tabbedPane = new JTabbedPane();

tabbedPane.setFont(new Font("Arial", Font.BOLD, 14));

tabbedPane.addTab("Resumen", crearPanelResumen());

tabbedPane.addTab("Detalle", crearPanelDetalle());

tabbedPane.addTab("Gráficos", crearPanelGraficos());

add(tabbedPane, BorderLayout.CENTER);

this.chartPanel = new ChartPanel(null);

this.chartPanel.setPreferredSize(new Dimension(700, 400));

this.chartPanel.setBorder(BorderFactory.createEtchedBorder());

add(crearPanelEstadisticas(), BorderLayout.SOUTH);

// ✅ Al final, cuando todo está inicializado

controlador.filtrarReporte();

}

private JPanel crearPanelControles() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));

// Panel de Filtros

JPanel panelFiltros = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 5));

panelFiltros.setBorder(BorderFactory.createTitledBorder("Filtrar Reporte"));

comboFiltro = new JComboBox<>(new String[]{"Todos", "Hoy", "Esta semana", "Este mes", "Rango personalizado"});

comboFiltro.setPreferredSize(new Dimension(150, 30));

dateChooserInicio = new JDateChooser();

dateChooserInicio.setPreferredSize(new Dimension(120, 30));

dateChooserInicio.setEnabled(false);

dateChooserFin = new JDateChooser();

dateChooserFin.setPreferredSize(new Dimension(120, 30));

dateChooserFin.setEnabled(false);

comboFiltro.addActionListener(e -> {

boolean rangoPersonalizado = "Rango personalizado".equals(comboFiltro.getSelectedItem());

dateChooserInicio.setEnabled(rangoPersonalizado);

dateChooserFin.setEnabled(rangoPersonalizado);

});

btnFiltrar = new JButton("Filtrar");

btnFiltrar.setPreferredSize(new Dimension(100, 30));

btnFiltrar.addActionListener(e -> controlador.filtrarReporte());

panelFiltros.add(new JLabel("Período:"));

panelFiltros.add(comboFiltro);

panelFiltros.add(new JLabel("Desde:"));

panelFiltros.add(dateChooserInicio);

panelFiltros.add(new JLabel("Hasta:"));

panelFiltros.add(dateChooserFin);

panelFiltros.add(btnFiltrar);

// Panel de Exportación

JPanel panelExportar = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 5));

panelExportar.setBorder(BorderFactory.createTitledBorder("Exportar Reporte"));

comboExportar = new JComboBox<>(new String[]{"PDF", "Excel", "HTML", "CSV"});

comboExportar.setPreferredSize(new Dimension(100, 30));

btnExportar = new JButton("Exportar");

btnExportar.setPreferredSize(new Dimension(100, 30));

btnExportar.addActionListener(e -> controlador.exportarReporte());

btnImprimir = new JButton("Imprimir");

btnImprimir.setPreferredSize(new Dimension(100, 30));

btnImprimir.addActionListener(e -> controlador.imprimirReporte());

panelExportar.add(new JLabel("Formato:"));

panelExportar.add(comboExportar);

panelExportar.add(btnExportar);

panelExportar.add(btnImprimir);

panel.add(panelFiltros);

panel.add(panelExportar);

return panel;

}

private JPanel crearPanelResumen() {

JPanel panel = new JPanel(new BorderLayout());

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Crear dataset vacío inicial

DefaultCategoryDataset dataset = new DefaultCategoryDataset();

// Crear gráfico inicial bien configurado

JFreeChart chart = ChartFactory.createBarChart(

"Resumen de Ventas",

"Fecha",

"Monto ($)",

dataset,

PlotOrientation.VERTICAL,

true, // Mostrar leyenda

true, // Mostrar tooltips

false // No URLs

);

// Configuración inicial del gráfico

CategoryPlot plot = chart.getCategoryPlot();

plot.setBackgroundPaint(Color.WHITE);

plot.setRangeGridlinePaint(Color.LIGHT\_GRAY);

// Configurar el renderizador

BarRenderer renderer = (BarRenderer) plot.getRenderer();

renderer.setSeriesPaint(0, new Color(79, 129, 189));

// Crear ChartPanel con el gráfico configurado

this.chartPanel = new ChartPanel(chart) {

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

// Solo mostrar mensaje si no hay datos

if (((CategoryPlot)getChart().getPlot()).getDataset().getRowCount() == 0) {

drawPlaceholder(g, "Esperando datos para mostrar el gráfico...");

}

}

};

// Configuración del panel

this.chartPanel.setPreferredSize(new Dimension(800, 500));

this.chartPanel.setMinimumSize(new Dimension(400, 300));

this.chartPanel.setBorder(BorderFactory.createEtchedBorder());

this.chartPanel.setBackground(Color.WHITE);

panel.add(this.chartPanel, BorderLayout.CENTER);

return panel;

}

// gráficos secundarios

public ChartPanel getGraficoMetodosPago() {

if (this.graficoMetodosPanel == null) {

initializeMetodosPagoPanel();

}

return this.graficoMetodosPanel;

}

public ChartPanel getGraficoProductos() {

if (this.graficoProductosPanel == null) {

initializeProductosPanel();

}

return this.graficoProductosPanel;

}

private void initializeMetodosPagoPanel() {

this.graficoMetodosPanel = new ChartPanel(null) {

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

if (getChart() == null) {

drawPlaceholder(g, "Gráfico de Métodos de Pago");

}

}

};

this.graficoMetodosPanel.setPreferredSize(new Dimension(350, 300));

this.graficoMetodosPanel.setBorder(BorderFactory.createTitledBorder("Métodos de Pago"));

}

private void initializeProductosPanel() {

this.graficoProductosPanel = new ChartPanel(null) {

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

if (getChart() == null) {

drawPlaceholder(g, "Gráfico de Productos");

}

}

};

this.graficoProductosPanel.setPreferredSize(new Dimension(350, 300));

this.graficoProductosPanel.setBorder(BorderFactory.createTitledBorder("Productos Más Vendidos"));

}

// Método para actualizar el resumen

public void actualizarResumen(List<Venta> ventas) {

System.out.println("[DEBUG] Actualizando resumen con " + ventas.size() + " ventas");

if (ventas == null || ventas.isEmpty()) {

mostrarMensajeEnGrafico("No hay datos para mostrar");

return;

}

try {

// 1. Preparar dataset con estructura adecuada

DefaultCategoryDataset dataset = new DefaultCategoryDataset();

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM HH:mm");

// 2. Procesar datos - usar fecha como categoría y "Ventas" como serie

Map<String, Double> ventasPorFecha = new LinkedHashMap<>();

for (Venta venta : ventas) {

String fechaHora = sdf.format(venta.getFecha());

ventasPorFecha.merge(fechaHora, venta.getTotal(), Double::sum);

}

// Agregar datos al dataset

ventasPorFecha.forEach((fecha, total) -> {

dataset.addValue(total, "Ventas", fecha);

});

System.out.println("[DEBUG] Dataset preparado con " + dataset.getRowCount() +

" filas y " + dataset.getColumnCount() + " columnas");

// 3. Obtener el gráfico existente o crear uno nuevo

JFreeChart chart = chartPanel.getChart();

if (chart == null) {

chart = ChartFactory.createBarChart(

"Resumen de Ventas",

"Fecha y Hora",

"Monto ($)",

dataset,

PlotOrientation.VERTICAL,

true, true, false

);

chartPanel.setChart(chart);

}

// 4. Actualizar el gráfico

CategoryPlot plot = (CategoryPlot) chart.getPlot();

plot.setDataset(dataset);

// Personalización del gráfico

BarRenderer renderer = (BarRenderer) plot.getRenderer();

renderer.setSeriesPaint(0, new Color(79, 129, 189)); // Color azul

renderer.setShadowVisible(true);

renderer.setShadowPaint(new Color(100, 100, 100));

// Configurar ejes para mejor visualización

CategoryAxis domainAxis = plot.getDomainAxis();

domainAxis.setCategoryLabelPositions(

CategoryLabelPositions.createUpRotationLabelPositions(Math.PI / 4.0) // Rotación de 45°

);

NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();

rangeAxis.setNumberFormatOverride(NumberFormat.getCurrencyInstance());

// Ajustar el rango del eje Y automáticamente

rangeAxis.setAutoRange(true);

// Forzar actualización visual

chartPanel.revalidate();

chartPanel.repaint();

} catch (Exception e) {

System.err.println("[ERROR] Al actualizar gráfico: " + e.getMessage());

e.printStackTrace();

mostrarError("Error al generar gráfico: " + e.getMessage());

}

}

// Modificar drawPlaceholder() para que no interfiera con el gráfico

private void drawPlaceholder(Graphics g, String message) {

// Verificar que realmente no hay datos

if (chartPanel != null && chartPanel.getChart() != null) {

CategoryPlot plot = (CategoryPlot) chartPanel.getChart().getPlot();

if (plot.getDataset() != null && plot.getDataset().getRowCount() > 0) {

return; // Hay datos, no mostrar placeholder

}

}

Graphics2D g2d = (Graphics2D) g.create();

try {

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

g2d.setFont(new Font("Arial", Font.ITALIC, 16));

g2d.setColor(new Color(100, 100, 100, 150)); // Gris semi-transparente

FontMetrics fm = g2d.getFontMetrics();

int x = (getWidth() - fm.stringWidth(message)) / 2;

int y = (getHeight() - fm.getHeight()) / 2 + fm.getAscent();

g2d.drawString(message, x, y);

} finally {

g2d.dispose();

}

}

private void mostrarMensajeEnGrafico(String mensaje) {

if (chartPanel != null) {

// Crear un gráfico vacío con el mensaje

JFreeChart chart = ChartFactory.createBarChart(

"Resumen de Ventas",

"",

"",

new DefaultCategoryDataset(),

PlotOrientation.VERTICAL,

false, false, false

);

chart.addSubtitle(new TextTitle(mensaje, new Font("Arial", Font.ITALIC, 14)));

chartPanel.setChart(chart);

chartPanel.repaint();

}

}

// Métodos auxiliares desglosados:

private DefaultCategoryDataset crearDatasetVentas(List<Venta> ventas) {

DefaultCategoryDataset dataset = new DefaultCategoryDataset();

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM");

Map<String, Double> ventasPorDia = new TreeMap<>(); // TreeMap para ordenar por fecha

for (Venta venta : ventas) {

String dia = sdf.format(venta.getFecha());

ventasPorDia.merge(dia, venta.getTotal(), Double::sum);

}

ventasPorDia.forEach((dia, total) -> {

dataset.addValue(total, "Ventas", dia);

});

return dataset;

}

private JFreeChart obtenerGraficoResumen(DefaultCategoryDataset dataset) {

if (graficoResumenPanel == null || graficoResumenPanel.getChart() == null) {

return ChartFactory.createBarChart(

"Resumen de Ventas Diarias",

"Fecha",

"Monto ($)",

dataset,

PlotOrientation.VERTICAL,

true, true, false

);

}

// Reutilizar gráfico existente

JFreeChart chart = graficoResumenPanel.getChart();

chart.getCategoryPlot().setDataset(dataset);

return chart;

}

private void personalizarGraficoResumen(JFreeChart chart) {

chart.setBackgroundPaint(Color.WHITE);

CategoryPlot plot = chart.getCategoryPlot();

plot.setBackgroundPaint(Color.WHITE);

plot.setRangeGridlinePaint(Color.LIGHT\_GRAY);

// Personalizar barras

BarRenderer renderer = (BarRenderer) plot.getRenderer();

renderer.setSeriesPaint(0, new Color(79, 129, 189)); // Azul corporativo

renderer.setShadowVisible(true);

renderer.setShadowPaint(new Color(200, 200, 200));

// Personalizar ejes

plot.getDomainAxis().setCategoryLabelPositions(

CategoryLabelPositions.createUpRotationLabelPositions(Math.PI / 6.0) // Rotar etiquetas 30°

);

NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();

rangeAxis.setNumberFormatOverride(NumberFormat.getCurrencyInstance());

}

private void actualizarPanelGrafico(JFreeChart chart) {

if (graficoResumenPanel == null) {

graficoResumenPanel = new ChartPanel(chart) {

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

if (getChart() == null) {

drawPlaceholder(g, "Esperando datos...");

}

}

};

graficoResumenPanel.setPreferredSize(new Dimension(700, 400));

graficoResumenPanel.setBorder(BorderFactory.createEtchedBorder());

} else {

graficoResumenPanel.setChart(chart);

}

graficoResumenPanel.repaint();

}

private void mostrarErrorEnGraficoResumen(String mensaje) {

DefaultCategoryDataset emptyDataset = new DefaultCategoryDataset();

JFreeChart errorChart = ChartFactory.createBarChart(

"Error en Resumen de Ventas",

"",

"",

emptyDataset,

PlotOrientation.VERTICAL,

false, false, false

);

errorChart.addSubtitle(new TextTitle(mensaje,

new Font("Arial", Font.PLAIN, 12)));

if (graficoResumenPanel != null) {

graficoResumenPanel.setChart(errorChart);

} else {

System.err.println("Error: graficoResumenPanel no está inicializado");

}

}

private JFreeChart createEmptyChart() {

JFreeChart chart = ChartFactory.createBarChart(

"Resumen de Ventas - Sin Datos",

"Período",

"Monto ($)",

new DefaultCategoryDataset()

);

TextTitle message = new TextTitle("No hay datos disponibles para el período seleccionado",

new Font("Arial", Font.ITALIC, 14));

chart.addSubtitle(message);

return chart;

}

private Map<String, Double> agruparVentasPorPeriodo(List<Venta> ventas) {

Map<String, Double> ventasAgrupadas = new LinkedHashMap<>();

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM");

for (Venta venta : ventas) {

String periodo = sdf.format(venta.getFecha());

ventasAgrupadas.merge(periodo, venta.getTotal(), Double::sum);

}

return ventasAgrupadas;

}

private void customizeChart(JFreeChart chart) {

CategoryPlot plot = chart.getCategoryPlot();

// Gradiente para el fondo

plot.setBackgroundPaint(new GradientPaint(

0, 0, new Color(240, 240, 240),

0, 100, new Color(200, 200, 200))

);

// Sombras en las barras

BarRenderer renderer = (BarRenderer) plot.getRenderer();

renderer.setShadowVisible(true);

renderer.setShadowPaint(Color.GRAY);

renderer.setSeriesPaint(0, new Color(44, 160, 44)); // Verde moderno

// Eje Y con formato monetario

NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();

rangeAxis.setNumberFormatOverride(NumberFormat.getCurrencyInstance());

// Rotar etiquetas del eje X

CategoryAxis domainAxis = plot.getDomainAxis();

domainAxis.setCategoryLabelPositions(

CategoryLabelPositions.createUpRotationLabelPositions(Math.PI / 6.0)

);

}

private JPanel crearPanelDetalle() {

JPanel panel = new JPanel(new BorderLayout());

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

txtAreaVentas = new JTextArea();

txtAreaVentas.setEditable(false);

txtAreaVentas.setFont(new Font("Monospaced", Font.PLAIN, 14));

JScrollPane scrollPane = new JScrollPane(txtAreaVentas);

panel.add(scrollPane, BorderLayout.CENTER);

// Mensaje inicial

txtAreaVentas.setText("Seleccione un período y haga clic en 'Filtrar' para ver el detalle de ventas");

return panel;

}

private JPanel crearPanelGraficos() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Inicializar paneles de gráficos

this.graficoMetodosPanel = new ChartPanel(null);

this.graficoProductosPanel = new ChartPanel(null);

// Configurar gráfico de métodos de pago

graficoMetodosPanel.setPreferredSize(new Dimension(350, 300));

graficoMetodosPanel.setBorder(BorderFactory.createTitledBorder("Métodos de Pago"));

// Configurar gráfico de productos

graficoProductosPanel.setPreferredSize(new Dimension(350, 300));

graficoProductosPanel.setBorder(BorderFactory.createTitledBorder("Productos Más Vendidos"));

panel.add(graficoMetodosPanel);

panel.add(graficoProductosPanel);

return panel;

}

private JPanel crearPanelEstadisticas() {

JPanel panel = new JPanel(new GridLayout(1, 4, 10, 0));

panel.setBorder(BorderFactory.createEmptyBorder(10, 0, 0, 0));

statsCards = new StatCard[4];

statsCards[0] = new StatCard("Total Ventas", "$0.00", new Color(144, 238, 144));

statsCards[1] = new StatCard("Ventas Hoy", "0", new Color(135, 206, 250));

statsCards[2] = new StatCard("Productos Vendidos", "0", new Color(255, 182, 193));

statsCards[3] = new StatCard("Ticket Promedio", "$0.00", new Color(221, 160, 221));

for (StatCard card : statsCards) {

card.setPreferredSize(new Dimension(250, 100));

panel.add(card);

}

return panel;

}

// Métodos para actualizar datos desde el controlador

public void mostrarDetalleVentas(List<Venta> ventas) {

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

StringBuilder sb = new StringBuilder();

sb.append("══════════════════════════════════════════════════════════════════════════════════\n");

sb.append(" DETALLE DE VENTAS\n");

sb.append("══════════════════════════════════════════════════════════════════════════════════\n");

sb.append(String.format("%-10s %-20s %-30s %-10s %-10s\n",

"ID Venta", "Fecha", "Producto", "Cantidad", "Total"));

sb.append("──────────────────────────────────────────────────────────────────────────────────\n");

for (Venta venta : ventas) {

for (Producto producto : venta.getProductos()) {

sb.append(String.format("%-10d %-20s %-30s %-10d $%-10.2f\n",

venta.getId(),

sdf.format(venta.getFecha()),

producto.getNombre(),

producto.getCantidad(),

producto.getPrecioUnitario() \* producto.getCantidad()));

}

}

sb.append("══════════════════════════════════════════════════════════════════════════════════\n");

sb.append(String.format("%62s $%-10.2f\n", "TOTAL VENTAS:",

ventas.stream().mapToDouble(Venta::getTotal).sum()));

txtAreaVentas.setText(sb.toString());

txtAreaVentas.setCaretPosition(0);

}

public void actualizarEstadisticas(String totalVentas, int ventasHoy, int productosVendidos, String ticketPromedio) {

statsCards[0].setValue(totalVentas);

statsCards[1].setValue(String.valueOf(ventasHoy));

statsCards[2].setValue(String.valueOf(productosVendidos));

statsCards[3].setValue(ticketPromedio);

for (StatCard card : statsCards) {

card.highlightChange();

}

}

public Date getFechaInicio() {

if ("Rango personalizado".equals(comboFiltro.getSelectedItem())) {

return dateChooserInicio.getDate();

} else {

Calendar cal = Calendar.getInstance();

switch (comboFiltro.getSelectedItem().toString()) {

case "Hoy": return cal.getTime();

case "Esta semana": cal.add(Calendar.DAY\_OF\_YEAR, -7); return cal.getTime();

case "Este mes": cal.add(Calendar.MONTH, -1); return cal.getTime();

default: return null; // "Todos"

}

}

}

public Date getFechaFin() {

if ("Rango personalizado".equals(comboFiltro.getSelectedItem())) {

return dateChooserFin.getDate();

}

return new Date(); // Fecha actual

}

public String getFormatoExportacion() {

return comboExportar.getSelectedItem().toString();

}

// Clase interna para las tarjetas de estadísticas

private class StatCard extends JPanel {

private JLabel valueLabel;

private Color originalColor;

public StatCard(String title, String initialValue, Color color) {

setLayout(new BorderLayout());

setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(color.darker(), 2),

BorderFactory.createEmptyBorder(15, 15, 15, 15)

));

setBackground(color.brighter());

this.originalColor = color.brighter();

JLabel titleLabel = new JLabel(title, SwingConstants.CENTER);

titleLabel.setFont(new Font("Arial", Font.BOLD, 14));

valueLabel = new JLabel(initialValue, SwingConstants.CENTER);

valueLabel.setFont(new Font("Arial", Font.BOLD, 18));

add(titleLabel, BorderLayout.NORTH);

add(valueLabel, BorderLayout.CENTER);

}

public void setValue(String value) {

valueLabel.setText(value);

}

public void highlightChange() {

Timer timer = new Timer(300, e -> setBackground(originalColor));

setBackground(Color.YELLOW);

timer.setRepeats(false);

timer.start();

}

}

public void mostrarError(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Error", JOptionPane.ERROR\_MESSAGE);

}

public void mostrarMensaje(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Información", JOptionPane.INFORMATION\_MESSAGE);

}

public void setControlador(ReportesControlador controlador) {

btnExportar.addActionListener(e -> controlador.exportarReporte());

btnImprimir.addActionListener(e -> controlador.imprimirReporte());

btnFiltrar.addActionListener(e -> controlador.filtrarReporte());

}

}

Una captura de pantalla de una red social

El contenido generado por IA puede ser incorrecto.

package Vista;

import Modelo.Usuario;

import Modelo.Producto;

import Controlador.ReportesControlador;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionListener;

import java.util.List;

import java.util.Map;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.data.category.DefaultCategoryDataset;

import org.jfree.data.general.DefaultPieDataset;

public class ReporteInventarioPanel extends JPanel {

private final ReportesControlador controlador;

private final CardLayout cardLayout;

private final JPanel panelContenido;

private JTable tablaInventario;

private JComboBox<String> comboFiltro;

private JComboBox<String> comboExportar;

private JLabel lblTotal, lblBajoStock, lblSinStock;

private ChartPanel chartPanelStock, chartPanelCategorias;

public ReporteInventarioPanel(Usuario usuario, ReportesControlador controlador,

CardLayout cardLayout, JPanel panelContenido) {

this.controlador = controlador;

this.cardLayout = cardLayout;

this.panelContenido = panelContenido;

controlador.setPanelInventario(this);

initUI();

cargarDatosIniciales();

}

private void initUI() {

setLayout(new BorderLayout(10, 10));

setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// 1. Panel de controles superiores (filtros y exportación)

JPanel panelControles = crearPanelControles();

add(panelControles, BorderLayout.NORTH);

// 2. Panel central con tabla de inventario y gráficas

JPanel panelCentral = new JPanel(new BorderLayout(10, 10));

// Tabla de inventario

JScrollPane scrollPane = crearTablaInventario();

panelCentral.add(scrollPane, BorderLayout.CENTER);

// Panel de gráficas

JPanel panelGraficas = crearPanelGraficas();

panelCentral.add(panelGraficas, BorderLayout.SOUTH);

add(panelCentral, BorderLayout.CENTER);

// 3. Panel inferior con botones

JPanel panelInferior = crearPanelInferior();

add(panelInferior, BorderLayout.SOUTH);

}

private JPanel crearPanelControles() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));

// Panel de Filtros

JPanel panelFiltros = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 5));

panelFiltros.setBorder(BorderFactory.createTitledBorder("Filtrar Inventario"));

comboFiltro = new JComboBox<>(new String[]{"Todos", "Bajo Stock", "Sobre Stock", "Categoría", "Proveedor"});

comboFiltro.setPreferredSize(new Dimension(150, 30));

JButton btnFiltrar = new JButton("Filtrar");

btnFiltrar.setPreferredSize(new Dimension(100, 30));

btnFiltrar.addActionListener(e -> controlador.filtrarInventario(

comboFiltro.getSelectedItem().toString()));

panelFiltros.add(new JLabel("Filtro:"));

panelFiltros.add(comboFiltro);

panelFiltros.add(btnFiltrar);

// Panel de Exportación

JPanel panelExportar = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 5));

panelExportar.setBorder(BorderFactory.createTitledBorder("Exportar Reporte"));

comboExportar = new JComboBox<>(new String[]{"PDF", "Excel", "HTML"});

comboExportar.setPreferredSize(new Dimension(100, 30));

JButton btnExportar = new JButton("Exportar");

btnExportar.setPreferredSize(new Dimension(100, 30));

btnExportar.addActionListener(e -> controlador.exportarReporteInventario(

comboExportar.getSelectedItem().toString()));

panelExportar.add(new JLabel("Formato:"));

panelExportar.add(comboExportar);

panelExportar.add(btnExportar);

panel.add(panelFiltros);

panel.add(panelExportar);

return panel;

}

private JScrollPane crearTablaInventario() {

// Columnas de la tabla

String[] columnNames = {"ID", "Producto", "Categoría", "Stock", "Stock Mín", "Stock Máx", "Precio", "Proveedor"};

// Modelo de tabla vacío inicialmente

tablaInventario = new JTable(new Object[0][columnNames.length], columnNames);

tablaInventario.setFont(new Font("Arial", Font.PLAIN, 12));

tablaInventario.getTableHeader().setFont(new Font("Arial", Font.BOLD, 12));

return new JScrollPane(tablaInventario);

}

private JPanel crearPanelGraficas() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createTitledBorder("Estadísticas de Inventario"));

// Gráfica 1: Niveles de stock

DefaultCategoryDataset datasetStock = new DefaultCategoryDataset();

datasetStock.addValue(0, "Stock", "Actual");

datasetStock.addValue(0, "Stock", "Mínimo");

datasetStock.addValue(0, "Stock", "Máximo");

JFreeChart chartStock = ChartFactory.createBarChart(

"Niveles de Stock Promedio",

"",

"Cantidad",

datasetStock

);

chartPanelStock = new ChartPanel(chartStock);

panel.add(chartPanelStock);

// Gráfica 2: Distribución por categoría

DefaultPieDataset datasetCategorias = new DefaultPieDataset();

datasetCategorias.setValue("Sin datos", 1);

JFreeChart chartCategorias = ChartFactory.createPieChart(

"Distribución por Categoría",

datasetCategorias,

true, true, false

);

chartPanelCategorias = new ChartPanel(chartCategorias);

panel.add(chartPanelCategorias);

return panel;

}

private JPanel crearPanelInferior() {

JPanel panel = new JPanel(new BorderLayout());

// Estadísticas rápidas

JPanel panelStats = new JPanel(new GridLayout(1, 3, 10, 0));

panelStats.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

lblTotal = new JLabel("Total Productos: 0", SwingConstants.CENTER);

lblBajoStock = new JLabel("Bajo Stock: 0", SwingConstants.CENTER);

lblSinStock = new JLabel("Sin Stock: 0", SwingConstants.CENTER);

panelStats.add(lblTotal);

panelStats.add(lblBajoStock);

panelStats.add(lblSinStock);

// Botón de regreso

JButton btnRegresar = new JButton("Regresar al Menú de Reportes");

btnRegresar.addActionListener(e -> mostrarMenuPrincipalReportes());

panel.add(panelStats, BorderLayout.CENTER);

panel.add(btnRegresar, BorderLayout.SOUTH);

return panel;

}

private void cargarDatosIniciales() {

controlador.cargarDatosInventario();

}

public void mostrarMenuPrincipalReportes() {

cardLayout.show(panelContenido, "menu\_principal");

}

// Métodos para actualizar la vista desde el controlador

public void actualizarTablaInventario(List<Producto> productos) {

String[] columnNames = {"ID", "Producto", "Categoría", "Stock", "Stock Mín", "Stock Máx", "Precio", "Proveedor"};

Object[][] data = new Object[productos.size()][columnNames.length];

for (int i = 0; i < productos.size(); i++) {

Producto p = productos.get(i);

data[i][0] = p.getId();

data[i][1] = p.getNombre();

data[i][2] = p.getCategoria();

data[i][3] = p.getCantidadDisponible();

data[i][4] = p.getStockMinimo();

data[i][5] = p.getStockMaximo();

data[i][6] = String.format("$%.2f", p.getPrecioVenta());

data[i][7] = p.getProveedor();

}

tablaInventario.setModel(new javax.swing.table.DefaultTableModel(data, columnNames));

}

public void actualizarEstadisticas(int totalProductos, int bajoStock, int sinStock) {

lblTotal.setText("Total Productos: " + totalProductos);

lblBajoStock.setText("Bajo Stock: " + bajoStock);

lblSinStock.setText("Sin Stock: " + sinStock);

}

public void actualizarGraficas(double stockPromedio, double stockMinPromedio, double stockMaxPromedio,

Map<String, Integer> distribucionCategorias) {

// Actualizar gráfica de niveles de stock

DefaultCategoryDataset datasetStock = new DefaultCategoryDataset();

datasetStock.addValue(stockPromedio, "Stock", "Actual");

datasetStock.addValue(stockMinPromedio, "Stock", "Mínimo");

datasetStock.addValue(stockMaxPromedio, "Stock", "Máximo");

JFreeChart chartStock = ChartFactory.createBarChart(

"Niveles de Stock Promedio",

"",

"Cantidad",

datasetStock

);

chartPanelStock.setChart(chartStock);

// Actualizar gráfica de distribución por categoría

DefaultPieDataset datasetCategorias = new DefaultPieDataset();

distribucionCategorias.forEach(datasetCategorias::setValue);

JFreeChart chartCategorias = ChartFactory.createPieChart(

"Distribución por Categoría",

datasetCategorias,

true, true, false

);

chartPanelCategorias.setChart(chartCategorias);

}

}

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

package Vista;

import Modelo.Cliente;

import Modelo.Usuario;

import Controlador.ReportesControlador;

import javax.swing.\*;

import java.awt.\*;

import java.text.SimpleDateFormat;

import java.util.List;

import java.util.Map;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.data.category.DefaultCategoryDataset;

import org.jfree.data.general.DefaultPieDataset;

public class ReporteClientePanel extends JPanel {

private final ReportesControlador controlador;

private final CardLayout cardLayout;

private final JPanel panelContenido;

private JTable tablaClientes;

private JComboBox<String> comboFiltro;

private JComboBox<String> comboExportar;

private JLabel lblTotal, lblActivos, lblInactivos;

private ChartPanel chartPanelRegistros, chartPanelPuntos;

public ReporteClientePanel(Usuario usuario, ReportesControlador controlador,

CardLayout cardLayout, JPanel panelContenido) {

this.controlador = controlador;

this.cardLayout = cardLayout;

this.panelContenido = panelContenido;

initUI();

cargarDatosIniciales();

}

private void initUI() {

setLayout(new BorderLayout(10, 10));

setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// 1. Panel de controles superiores (filtros y exportación)

JPanel panelControles = crearPanelControles();

add(panelControles, BorderLayout.NORTH);

// 2. Panel central con tabla de clientes y gráficas

JPanel panelCentral = new JPanel(new BorderLayout(10, 10));

// Tabla de clientes

JScrollPane scrollPane = crearTablaClientes();

panelCentral.add(scrollPane, BorderLayout.CENTER);

// Panel de gráficas

JPanel panelGraficas = crearPanelGraficas();

panelCentral.add(panelGraficas, BorderLayout.SOUTH);

add(panelCentral, BorderLayout.CENTER);

// 3. Panel inferior con botones

JPanel panelInferior = crearPanelInferior();

add(panelInferior, BorderLayout.SOUTH);

}

private JPanel crearPanelControles() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));

// Panel de Filtros

JPanel panelFiltros = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 5));

panelFiltros.setBorder(BorderFactory.createTitledBorder("Filtrar Clientes"));

comboFiltro = new JComboBox<>(new String[]{"Todos", "Activos", "Inactivos", "Con Puntos", "Sin Puntos"});

comboFiltro.setPreferredSize(new Dimension(150, 30));

JButton btnFiltrar = new JButton("Filtrar");

btnFiltrar.setPreferredSize(new Dimension(100, 30));

btnFiltrar.addActionListener(e -> controlador.filtrarClientes(

comboFiltro.getSelectedItem().toString()));

panelFiltros.add(new JLabel("Filtro:"));

panelFiltros.add(comboFiltro);

panelFiltros.add(btnFiltrar);

// Panel de Exportación

JPanel panelExportar = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 5));

panelExportar.setBorder(BorderFactory.createTitledBorder("Exportar Reporte"));

comboExportar = new JComboBox<>(new String[]{"PDF", "Excel", "HTML"});

comboExportar.setPreferredSize(new Dimension(100, 30));

JButton btnExportar = new JButton("Exportar");

btnExportar.setPreferredSize(new Dimension(100, 30));

btnExportar.addActionListener(e -> controlador.exportarReporteClientes(

comboExportar.getSelectedItem().toString()));

panelExportar.add(new JLabel("Formato:"));

panelExportar.add(comboExportar);

panelExportar.add(btnExportar);

panel.add(panelFiltros);

panel.add(panelExportar);

return panel;

}

private JScrollPane crearTablaClientes() {

// Columnas de la tabla

String[] columnNames = {"ID", "Teléfono", "Nombre", "Última Compra", "Puntos", "Fecha Registro"};

// Modelo de tabla vacío inicialmente

tablaClientes = new JTable(new Object[0][columnNames.length], columnNames);

tablaClientes.setFont(new Font("Arial", Font.PLAIN, 12));

tablaClientes.getTableHeader().setFont(new Font("Arial", Font.BOLD, 12));

return new JScrollPane(tablaClientes);

}

private JPanel crearPanelGraficas() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createTitledBorder("Estadísticas de Clientes"));

// Gráfica 1: Registros por mes

DefaultCategoryDataset datasetRegistros = new DefaultCategoryDataset();

datasetRegistros.addValue(0, "Registros", "Ene");

datasetRegistros.addValue(0, "Registros", "Feb");

// ... otros meses

JFreeChart chartRegistros = ChartFactory.createBarChart(

"Registros de Clientes por Mes",

"Mes",

"Cantidad",

datasetRegistros

);

chartPanelRegistros = new ChartPanel(chartRegistros);

panel.add(chartPanelRegistros);

// Gráfica 2: Distribución de puntos

DefaultPieDataset datasetPuntos = new DefaultPieDataset();

datasetPuntos.setValue("0-100 pts", 0);

datasetPuntos.setValue("101-500 pts", 0);

datasetPuntos.setValue("501+ pts", 0);

JFreeChart chartPuntos = ChartFactory.createPieChart(

"Distribución de Puntos",

datasetPuntos,

true, true, false

);

chartPanelPuntos = new ChartPanel(chartPuntos);

panel.add(chartPanelPuntos);

return panel;

}

private JPanel crearPanelInferior() {

JPanel panel = new JPanel(new BorderLayout());

// Estadísticas rápidas

JPanel panelStats = new JPanel(new GridLayout(1, 3, 10, 0));

panelStats.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

lblTotal = new JLabel("Total Clientes: 0", SwingConstants.CENTER);

lblActivos = new JLabel("Activos: 0", SwingConstants.CENTER);

lblInactivos = new JLabel("Inactivos: 0", SwingConstants.CENTER);

panelStats.add(lblTotal);

panelStats.add(lblActivos);

panelStats.add(lblInactivos);

// Botón de regreso

JButton btnRegresar = new JButton("Regresar al Menú de Reportes");

btnRegresar.addActionListener(e -> mostrarMenuPrincipalReportes());

panel.add(panelStats, BorderLayout.CENTER);

panel.add(btnRegresar, BorderLayout.SOUTH);

return panel;

}

private void cargarDatosIniciales() {

controlador.cargarDatosClientes();

}

public void mostrarMenuPrincipalReportes() {

cardLayout.show(panelContenido, "menu\_principal");

}

// Métodos para actualizar la vista desde el controlador

public void actualizarTablaClientes(List<Cliente> clientes) {

String[] columnNames = {"ID", "Teléfono", "Nombre", "Última Compra", "Puntos", "Fecha Registro"};

Object[][] data = new Object[clientes.size()][columnNames.length];

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

for (int i = 0; i < clientes.size(); i++) {

Cliente c = clientes.get(i);

data[i][0] = c.getId();

data[i][1] = c.getTelefono();

data[i][2] = c.getNombre();

data[i][3] = c.getUltimaCompra();

data[i][4] = c.getPuntos();

data[i][5] = c.getFechaRegistro() != null ? sdf.format(c.getFechaRegistro()) : "";

}

tablaClientes.setModel(new javax.swing.table.DefaultTableModel(data, columnNames));

}

public void actualizarEstadisticas(int totalClientes, int activos, int inactivos) {

lblTotal.setText("Total Clientes: " + totalClientes);

lblActivos.setText("Activos: " + activos);

lblInactivos.setText("Inactivos: " + inactivos);

}

public void actualizarGraficas(Map<String, Integer> registrosPorMes,

Map<String, Integer> distribucionPuntos) {

// Actualizar gráfica de registros por mes

DefaultCategoryDataset datasetRegistros = new DefaultCategoryDataset();

registrosPorMes.forEach((mes, cantidad) -> {

datasetRegistros.addValue(cantidad, "Registros", mes);

});

JFreeChart chartRegistros = ChartFactory.createBarChart(

"Registros de Clientes por Mes",

"Mes",

"Cantidad",

datasetRegistros

);

chartPanelRegistros.setChart(chartRegistros);

// Actualizar gráfica de distribución de puntos

DefaultPieDataset datasetPuntos = new DefaultPieDataset();

distribucionPuntos.forEach(datasetPuntos::setValue);

JFreeChart chartPuntos = ChartFactory.createPieChart(

"Distribución de Puntos",

datasetPuntos,

true, true, false

);

chartPanelPuntos.setChart(chartPuntos);

}

public void mostrarMensaje(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Información", JOptionPane.INFORMATION\_MESSAGE);

}

public void mostrarError(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Error", JOptionPane.ERROR\_MESSAGE);

}

}

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

package Vista;

import Modelo.Usuario;

import Modelo.Proveedor;

import Controlador.ReportesControlador;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionListener;

import java.text.SimpleDateFormat;

import java.util.List;

import java.util.Map;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.data.category.DefaultCategoryDataset;

import org.jfree.data.general.DefaultPieDataset;

public class ReporteProveedoresPanel extends JPanel {

private final ReportesControlador controlador;

private final CardLayout cardLayout;

private final JPanel panelContenido;

private JTable tablaProveedores;

private JComboBox<String> comboFiltro;

private JComboBox<String> comboExportar;

private JLabel lblTotal, lblActivos, lblUltimaVisita;

private ChartPanel chartPanelVisitas, chartPanelProductos;

public ReporteProveedoresPanel(Usuario usuario, ReportesControlador controlador,

CardLayout cardLayout, JPanel panelContenido) {

this.controlador = controlador;

this.cardLayout = cardLayout;

this.panelContenido = panelContenido;

initUI();

cargarDatosIniciales();

}

private void initUI() {

setLayout(new BorderLayout(10, 10));

setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// 1. Panel de controles superiores (filtros y exportación)

JPanel panelControles = crearPanelControles();

add(panelControles, BorderLayout.NORTH);

// 2. Panel central con tabla de proveedores y gráficas

JPanel panelCentral = new JPanel(new BorderLayout(10, 10));

// Tabla de proveedores

JScrollPane scrollPane = crearTablaProveedores();

panelCentral.add(scrollPane, BorderLayout.CENTER);

// Panel de gráficas

JPanel panelGraficas = crearPanelGraficas();

panelCentral.add(panelGraficas, BorderLayout.SOUTH);

add(panelCentral, BorderLayout.CENTER);

// 3. Panel inferior con botones

JPanel panelInferior = crearPanelInferior();

add(panelInferior, BorderLayout.SOUTH);

}

private JPanel crearPanelControles() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createEmptyBorder(0, 0, 15, 0));

// Panel de Filtros

JPanel panelFiltros = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 5));

panelFiltros.setBorder(BorderFactory.createTitledBorder("Filtrar Proveedores"));

comboFiltro = new JComboBox<>(new String[]{"Todos", "Con Visita Reciente", "Sin Visita Reciente", "Por Producto"});

comboFiltro.setPreferredSize(new Dimension(180, 30));

JButton btnFiltrar = new JButton("Filtrar");

btnFiltrar.setPreferredSize(new Dimension(100, 30));

btnFiltrar.addActionListener(e -> controlador.filtrarProveedores(

comboFiltro.getSelectedItem().toString()));

panelFiltros.add(new JLabel("Filtro:"));

panelFiltros.add(comboFiltro);

panelFiltros.add(btnFiltrar);

// Panel de Exportación

JPanel panelExportar = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 5));

panelExportar.setBorder(BorderFactory.createTitledBorder("Exportar Reporte"));

comboExportar = new JComboBox<>(new String[]{"PDF", "Excel", "HTML"});

comboExportar.setPreferredSize(new Dimension(100, 30));

JButton btnExportar = new JButton("Exportar");

btnExportar.setPreferredSize(new Dimension(100, 30));

btnExportar.addActionListener(e -> controlador.exportarReporteProveedores(

comboExportar.getSelectedItem().toString()));

panelExportar.add(new JLabel("Formato:"));

panelExportar.add(comboExportar);

panelExportar.add(btnExportar);

panel.add(panelFiltros);

panel.add(panelExportar);

return panel;

}

private JScrollPane crearTablaProveedores() {

// Columnas de la tabla

String[] columnNames = {"ID", "Nombre", "Teléfono", "Dirección", "Producto Suministrado", "Última Visita"};

// Modelo de tabla vacío inicialmente

tablaProveedores = new JTable(new Object[0][columnNames.length], columnNames);

tablaProveedores.setFont(new Font("Arial", Font.PLAIN, 12));

tablaProveedores.getTableHeader().setFont(new Font("Arial", Font.BOLD, 12));

return new JScrollPane(tablaProveedores);

}

private JPanel crearPanelGraficas() {

JPanel panel = new JPanel(new GridLayout(1, 2, 10, 0));

panel.setBorder(BorderFactory.createTitledBorder("Estadísticas de Proveedores"));

// Gráfica 1: Visitas por mes

DefaultCategoryDataset datasetVisitas = new DefaultCategoryDataset();

datasetVisitas.addValue(0, "Visitas", "Ene");

datasetVisitas.addValue(0, "Visitas", "Feb");

datasetVisitas.addValue(0, "Visitas", "Mar");

JFreeChart chartVisitas = ChartFactory.createBarChart(

"Visitas por Mes",

"Mes",

"Número de Visitas",

datasetVisitas

);

chartPanelVisitas = new ChartPanel(chartVisitas);

panel.add(chartPanelVisitas);

// Gráfica 2: Distribución por producto suministrado

DefaultPieDataset datasetProductos = new DefaultPieDataset();

datasetProductos.setValue("Sin datos", 1);

JFreeChart chartProductos = ChartFactory.createPieChart(

"Distribución por Producto",

datasetProductos,

true, true, false

);

chartPanelProductos = new ChartPanel(chartProductos);

panel.add(chartPanelProductos);

return panel;

}

private JPanel crearPanelInferior() {

JPanel panel = new JPanel(new BorderLayout());

// Estadísticas rápidas

JPanel panelStats = new JPanel(new GridLayout(1, 3, 10, 0));

panelStats.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

lblTotal = new JLabel("Total Proveedores: 0", SwingConstants.CENTER);

lblActivos = new JLabel("Visitados este mes: 0", SwingConstants.CENTER);

lblUltimaVisita = new JLabel("Última visita: N/A", SwingConstants.CENTER);

panelStats.add(lblTotal);

panelStats.add(lblActivos);

panelStats.add(lblUltimaVisita);

// Botón de regreso

JButton btnRegresar = new JButton("Regresar al Menú de Reportes");

btnRegresar.addActionListener(e -> mostrarMenuPrincipalReportes());

panel.add(panelStats, BorderLayout.CENTER);

panel.add(btnRegresar, BorderLayout.SOUTH);

return panel;

}

private void cargarDatosIniciales() {

controlador.cargarDatosProveedores();

}

public void mostrarMenuPrincipalReportes() {

cardLayout.show(panelContenido, "menu\_principal");

}

// Métodos para actualizar la vista desde el controlador

public void actualizarTablaProveedores(List<Proveedor> proveedores) {

String[] columnNames = {"ID", "Nombre", "Teléfono", "Dirección", "Producto Suministrado", "Última Visita"};

Object[][] data = new Object[proveedores.size()][columnNames.length];

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (int i = 0; i < proveedores.size(); i++) {

Proveedor p = proveedores.get(i);

data[i][0] = p.getId();

data[i][1] = p.getNombre();

data[i][2] = p.getTelefono();

data[i][3] = p.getDireccion();

data[i][4] = p.getProductoSuministrado();

data[i][5] = p.getUltimaVisita() != null ? sdf.format(p.getUltimaVisita()) : "Nunca";

}

tablaProveedores.setModel(new javax.swing.table.DefaultTableModel(data, columnNames));

}

public void actualizarEstadisticas(int totalProveedores, int visitadosEsteMes, String ultimaVisita) {

lblTotal.setText("Total Proveedores: " + totalProveedores);

lblActivos.setText("Visitados este mes: " + visitadosEsteMes);

lblUltimaVisita.setText("Última visita: " + ultimaVisita);

}

public void actualizarGraficas(Map<String, Integer> visitasPorMes, Map<String, Integer> distribucionProductos) {

// Actualizar gráfica de visitas por mes

DefaultCategoryDataset datasetVisitas = new DefaultCategoryDataset();

visitasPorMes.forEach((mes, cantidad) -> {

datasetVisitas.addValue(cantidad, "Visitas", mes);

});

JFreeChart chartVisitas = ChartFactory.createBarChart(

"Visitas por Mes",

"Mes",

"Número de Visitas",

datasetVisitas

);

chartPanelVisitas.setChart(chartVisitas);

// Actualizar gráfica de distribución por producto

DefaultPieDataset datasetProductos = new DefaultPieDataset();

distribucionProductos.forEach(datasetProductos::setValue);

JFreeChart chartProductos = ChartFactory.createPieChart(

"Distribución por Producto",

datasetProductos,

true, true, false

);

chartPanelProductos.setChart(chartProductos);

}

}

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

package Vista;

import javax.swing.\*;

import javax.swing.border.\*;

import javax.swing.table.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

import java.sql.\*;

import java.text.\*;

import java.util.\*;

import java.util.Date;

import java.util.List;

import java.util.stream.Collectors;

import Modelo.Venta;

import Modelo.Producto;

import Modelo.Usuario;

import Modelo.Cliente;

import ConexionBD.ConexionAccess;

import Controlador.ReportesControlador;

public class ReimprimirDialog extends JDialog {

private JComboBox<String> cmbTipoDocumento;

private JTextField txtIdDocumento;

private JButton btnBuscar;

private JButton btnReimprimir;

private ReportesControlador controlador;

private Usuario usuario;

public ReimprimirDialog(Usuario usuario,Frame parent, ReportesControlador controlador) {

this.usuario = usuario;

super(parent, "Reimprimir Documento", true);

this.controlador = controlador;

initUI();

}

private void initUI() {

// Configuración básica del diálogo

setLayout(new BorderLayout(10, 10));

setSize(450, 200);

setResizable(false);

getContentPane().setBackground(new Color(245, 245, 245));

// Panel principal

JPanel mainPanel = new JPanel(new BorderLayout(10, 15));

mainPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

mainPanel.setBackground(new Color(245, 245, 245));

// Panel de controles

JPanel controlPanel = new JPanel(new GridLayout(2, 2, 10, 10));

controlPanel.setBackground(new Color(245, 245, 245));

// Componentes

JLabel lblTipo = new JLabel("Tipo de documento:");

lblTipo.setFont(new Font("Segoe UI", Font.PLAIN, 12));

cmbTipoDocumento = new JComboBox<>(new String[]{"Ticket de Venta", "Reporte de Ventas",

"Reporte de Inventario", "Reporte de Clientes", "Reporte de Proveedores"});

styleComboBox(cmbTipoDocumento);

JLabel lblId = new JLabel("ID o referencia:");

lblId.setFont(new Font("Segoe UI", Font.PLAIN, 12));

txtIdDocumento = new JTextField();

styleTextField(txtIdDocumento);

controlPanel.add(lblTipo);

controlPanel.add(cmbTipoDocumento);

controlPanel.add(lblId);

controlPanel.add(txtIdDocumento);

// Panel de botones

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 0));

buttonPanel.setBackground(new Color(245, 245, 245));

btnBuscar = createStyledButton("Buscar", new Color(70, 130, 180));

btnBuscar.addActionListener(e -> buscarDocumento());

btnReimprimir = createStyledButton("Reimprimir", new Color(46, 125, 50));

btnReimprimir.addActionListener(e -> reimprimirDocumento());

buttonPanel.add(btnBuscar);

buttonPanel.add(btnReimprimir);

mainPanel.add(controlPanel, BorderLayout.CENTER);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

add(mainPanel);

setLocationRelativeTo(getParent());

}

private void styleComboBox(JComboBox<String> combo) {

combo.setFont(new Font("Segoe UI", Font.PLAIN, 12));

combo.setBackground(Color.WHITE);

combo.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(200, 200, 200)),

BorderFactory.createEmptyBorder(5, 8, 5, 8)

));

}

private void styleTextField(JTextField textField) {

textField.setFont(new Font("Segoe UI", Font.PLAIN, 12));

textField.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(200, 200, 200)),

BorderFactory.createEmptyBorder(5, 8, 5, 8)

));

}

private JButton createStyledButton(String text, Color bgColor) {

JButton button = new JButton(text);

button.setFont(new Font("Segoe UI", Font.BOLD, 12));

button.setBackground(bgColor);

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createEmptyBorder(8, 15, 8, 15));

button.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

button.addMouseListener(new MouseAdapter() {

public void mouseEntered(MouseEvent e) {

button.setBackground(bgColor.darker());

}

public void mouseExited(MouseEvent e) {

button.setBackground(bgColor);

}

});

return button;

}

private void buscarDocumento() {

String tipoSeleccionado = (String) cmbTipoDocumento.getSelectedItem();

if (tipoSeleccionado.equals("Ticket de Venta")) {

mostrarDialogoBusquedaTickets();

} else {

String tipoReporte = tipoSeleccionado.replace("Reporte de ", "").toUpperCase();

mostrarDialogoBusquedaReportes(tipoReporte);

}

}

private void mostrarDialogoBusquedaTickets() {

JDialog searchDialog = new JDialog(this, "Buscar Tickets de Venta", true);

searchDialog.setLayout(new BorderLayout());

searchDialog.setSize(900, 600);

searchDialog.getContentPane().setBackground(new Color(245, 245, 245));

// Panel de filtros

JPanel filterPanel = new JPanel(new GridLayout(2, 4, 10, 10));

filterPanel.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createMatteBorder(0, 0, 1, 0, new Color(220, 220, 220)),

BorderFactory.createEmptyBorder(15, 15, 15, 15)

));

filterPanel.setBackground(Color.WHITE);

// Componentes de filtro

JTextField txtId = new JTextField();

styleTextField(txtId);

JSpinner spinnerDesde = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor editorDesde = new JSpinner.DateEditor(spinnerDesde, "dd/MM/yyyy");

spinnerDesde.setEditor(editorDesde);

spinnerDesde.setValue(new Date());

JSpinner spinnerHasta = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor editorHasta = new JSpinner.DateEditor(spinnerHasta, "dd/MM/yyyy");

spinnerHasta.setEditor(editorHasta);

spinnerHasta.setValue(new Date());

JComboBox<String> cmbPayment = new JComboBox<>(new String[]{"Todos", "EFECTIVO", "TARJETA"});

styleComboBox(cmbPayment);

// Agregar componentes al panel

filterPanel.add(createFilterLabel("ID Venta:"));

filterPanel.add(txtId);

filterPanel.add(createFilterLabel("Fecha Desde:"));

filterPanel.add(spinnerDesde);

filterPanel.add(createFilterLabel("Fecha Hasta:"));

filterPanel.add(spinnerHasta);

filterPanel.add(createFilterLabel("Método Pago:"));

filterPanel.add(cmbPayment);

// Tabla de resultados

String[] columnas = {"ID Venta", "Fecha", "Total", "Método Pago", "Cliente", "Productos"};

DefaultTableModel model = new DefaultTableModel(columnas, 0) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

};

JTable resultsTable = new JTable(model);

configurarTabla(resultsTable);

// Panel de botones

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 10));

buttonPanel.setBorder(BorderFactory.createEmptyBorder(5, 0, 5, 10));

buttonPanel.setBackground(Color.WHITE);

JButton btnExport = createStyledButton("Exportar", new Color(56, 142, 60));

btnExport.addActionListener(e -> exportarTablaACSV(resultsTable));

JButton btnSearch = createStyledButton("Buscar", new Color(51, 103, 214));

btnSearch.addActionListener(e -> realizarBusqueda(txtId, spinnerDesde, spinnerHasta, cmbPayment, resultsTable));

JButton btnSelect = createStyledButton("Seleccionar", new Color(103, 58, 183));

btnSelect.addActionListener(e -> seleccionarTicket(searchDialog, resultsTable));

JButton btnCancel = createStyledButton("Cancelar", new Color(158, 158, 158));

btnCancel.addActionListener(e -> searchDialog.dispose());

buttonPanel.add(btnExport);

buttonPanel.add(btnSearch);

buttonPanel.add(btnSelect);

buttonPanel.add(btnCancel);

searchDialog.add(filterPanel, BorderLayout.NORTH);

searchDialog.add(new JScrollPane(resultsTable), BorderLayout.CENTER);

searchDialog.add(buttonPanel, BorderLayout.SOUTH);

searchDialog.setLocationRelativeTo(this);

searchDialog.setVisible(true);

}

private JLabel createFilterLabel(String text) {

JLabel label = new JLabel(text);

label.setFont(new Font("Segoe UI", Font.BOLD, 12));

return label;

}

private void configurarTabla(JTable table) {

table.setFont(new Font("Segoe UI", Font.PLAIN, 12));

table.setRowHeight(25);

table.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

table.setAutoCreateRowSorter(true);

table.setGridColor(new Color(240, 240, 240));

table.setShowVerticalLines(false);

table.setDefaultRenderer(Object.class, new DefaultTableCellRenderer() {

@Override

public Component getTableCellRendererComponent(JTable table, Object value,

boolean isSelected, boolean hasFocus, int row, int column) {

super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

setBorder(noFocusBorder);

if (isSelected) {

setBackground(new Color(220, 240, 255));

} else {

setBackground(row % 2 == 0 ? Color.WHITE : new Color(248, 248, 248));

}

return this;

}

});

}

private void realizarBusqueda(JTextField txtId, JSpinner spinnerDesde, JSpinner spinnerHasta,

JComboBox<String> cmbPayment, JTable resultsTable) {

DefaultTableModel model = (DefaultTableModel) resultsTable.getModel();

model.setRowCount(0);

try {

String idVenta = txtId.getText().trim();

Date fechaDesde = (Date) spinnerDesde.getValue();

Date fechaHasta = (Date) spinnerHasta.getValue();

String metodoPago = cmbPayment.getSelectedIndex() == 0 ? null : cmbPayment.getSelectedItem().toString();

// Ajustar fecha hasta para incluir todo el día

Calendar cal = Calendar.getInstance();

cal.setTime(fechaHasta);

cal.set(Calendar.HOUR\_OF\_DAY, 23);

cal.set(Calendar.MINUTE, 59);

cal.set(Calendar.SECOND, 59);

fechaHasta = cal.getTime();

List<Venta> ventas = buscarVentasEnBD(idVenta, fechaDesde, fechaHasta, metodoPago);

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (Venta venta : ventas) {

String cliente = venta.getCliente() != null ? venta.getCliente().getNombre() : "N/A";

String productos = venta.getProductos().stream()

.map(p -> p.getNombre())

.collect(Collectors.joining(", "));

model.addRow(new Object[]{

venta.getId(),

sdf.format(venta.getFecha()),

String.format("$%,.2f", venta.getTotal()),

venta.getMetodoPago(),

cliente,

productos.length() > 50 ? productos.substring(0, 47) + "..." : productos

});

}

} catch (Exception e) {

JOptionPane.showMessageDialog(this, "Error en la búsqueda: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

private List<Venta> buscarVentasEnBD(String idVenta, Date fechaDesde, Date fechaHasta, String metodoPago) {

List<Venta> ventas = new ArrayList<>();

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

conn = ConexionAccess.conectar();

StringBuilder sql = new StringBuilder(

"SELECT v.id, v.fecha, v.total, v.metodo\_pago, c.nombre as cliente " +

"FROM Ventas v " +

"LEFT JOIN Clientes c ON v.id\_cliente = c.id " +

"WHERE 1=1");

if (!idVenta.isEmpty()) {

sql.append(" AND v.id = ?");

}

if (fechaDesde != null) {

sql.append(" AND v.fecha >= ?");

}

if (fechaHasta != null) {

sql.append(" AND v.fecha <= ?");

}

if (metodoPago != null) {

sql.append(" AND v.metodo\_pago = ?");

}

sql.append(" ORDER BY v.fecha DESC");

pstmt = conn.prepareStatement(sql.toString());

int paramIndex = 1;

if (!idVenta.isEmpty()) {

pstmt.setString(paramIndex++, idVenta);

}

if (fechaDesde != null) {

pstmt.setTimestamp(paramIndex++, new Timestamp(fechaDesde.getTime()));

}

if (fechaHasta != null) {

pstmt.setTimestamp(paramIndex++, new Timestamp(fechaHasta.getTime()));

}

if (metodoPago != null) {

pstmt.setString(paramIndex, metodoPago);

}

rs = pstmt.executeQuery();

while (rs.next()) {

Venta venta = new Venta();

venta.setId(rs.getInt("id"));

venta.setFecha(rs.getTimestamp("fecha"));

venta.setTotal(rs.getDouble("total"));

venta.setMetodoPago(rs.getString("metodo\_pago"));

if (rs.getString("cliente") != null) {

Cliente cliente = new Cliente();

cliente.setNombre(rs.getString("cliente"));

venta.setCliente(cliente);

}

venta.setProductos(obtenerProductosVenta(conn, venta.getId()));

ventas.add(venta);

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

try {

if (rs != null) rs.close();

if (pstmt != null) pstmt.close();

if (conn != null) conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

return ventas;

}

private List<Producto> obtenerProductosVenta(Connection conn, int idVenta) throws SQLException {

List<Producto> productos = new ArrayList<>();

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

String sql = "SELECT p.id, p.nombre, dv.cantidad, dv.precio\_unitario " +

"FROM DetalleVenta dv " +

"JOIN Productos p ON dv.id\_producto = p.id " +

"WHERE dv.id\_venta = ?";

pstmt = conn.prepareStatement(sql);

pstmt.setInt(1, idVenta);

rs = pstmt.executeQuery();

while (rs.next()) {

Producto producto = new Producto(usuario);

producto.setId(rs.getString("id"));

producto.setNombre(rs.getString("nombre"));

producto.setCantidad(rs.getInt("cantidad"));

producto.setPrecioUnitario(rs.getDouble("precio\_unitario"));

productos.add(producto);

}

} finally {

if (rs != null) rs.close();

if (pstmt != null) pstmt.close();

}

return productos;

}

private void exportarTablaACSV(JTable table) {

JFileChooser fileChooser = new JFileChooser();

fileChooser.setDialogTitle("Guardar como CSV");

fileChooser.setSelectedFile(new File("tickets\_exportados.csv"));

if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE\_OPTION) {

try (PrintWriter pw = new PrintWriter(fileChooser.getSelectedFile())) {

// Encabezados

for (int i = 0; i < table.getColumnCount(); i++) {

pw.print(table.getColumnName(i));

if (i < table.getColumnCount() - 1) pw.print(",");

}

pw.println();

// Datos

for (int row = 0; row < table.getRowCount(); row++) {

for (int col = 0; col < table.getColumnCount(); col++) {

Object value = table.getValueAt(row, col);

pw.print(value != null ? value.toString().replace(",", ";") : "");

if (col < table.getColumnCount() - 1) pw.print(",");

}

pw.println();

}

JOptionPane.showMessageDialog(this, "Datos exportados exitosamente");

} catch (IOException e) {

JOptionPane.showMessageDialog(this, "Error al exportar: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

private void seleccionarTicket(JDialog parent, JTable table) {

int row = table.getSelectedRow();

if (row >= 0) {

String id = table.getModel().getValueAt(row, 0).toString();

txtIdDocumento.setText(id);

parent.dispose();

} else {

JOptionPane.showMessageDialog(parent,

"Por favor seleccione un ticket de la lista",

"Advertencia", JOptionPane.WARNING\_MESSAGE);

}

}

private void mostrarDialogoBusquedaReportes(String tipoReporte) {

String idReporte = JOptionPane.showInputDialog(

this,

"Ingrese el ID del reporte de " + tipoReporte,

"Buscar Reporte",

JOptionPane.QUESTION\_MESSAGE

);

if (idReporte != null && !idReporte.trim().isEmpty()) {

try {

String reportType = "Reporte de " + tipoReporte; // Ej: "Reporte de Clientes"

controlador.reimprimirReporte(reportType.toUpperCase(), idReporte.trim());

} catch (Exception e) {

JOptionPane.showMessageDialog(this, "Error al buscar: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

private void reimprimirDocumento() {

String type = (String) cmbTipoDocumento.getSelectedItem();

String id = txtIdDocumento.getText().trim();

System.out.println("[DEBUG] Intentando reimprimir: Tipo=" + type + ", ID=" + id); // Log

if (id.isEmpty()) {

JOptionPane.showMessageDialog(this, "Por favor ingrese un ID válido",

"Error", JOptionPane.ERROR\_MESSAGE);

return;

}

try {

if (type.equals("Ticket de Venta")) {

System.out.println("[DEBUG] Reimprimiendo ticket...");

controlador.reimprimirTicketVenta(id);

} else {

String reportType = type.replace("Reporte de ", "").toUpperCase();

System.out.println("[DEBUG] Reimprimiendo reporte: " + reportType);

controlador.reimprimirReporte(reportType, id);

}

} catch (Exception e) {

System.err.println("[ERROR] Fallo al reimprimir: " + e.getMessage()); // Log

JOptionPane.showMessageDialog(this, "Error al reimprimir: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

# Inventario

package Modelo;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Date;

import java.util.List;

import ConexionBD.ConexionAccess;

public class Inventario {

private InventarioDAO inventarioDAO;

private List<String> categorias;

public Inventario() {

this.inventarioDAO = new InventarioDAO();

inicializarCategorias();

}

private void inicializarCategorias() {

categorias = new ArrayList<>();

categorias.add("TODOS");

categorias.add("Abarrotes");

categorias.add("Panaderia y Tortilleria");

categorias.add("Carnes y embutidos");

categorias.add("Botanas y dulces");

categorias.add("Bebidas");

categorias.add("Lacteos");

categorias.add("Limpieza del hogar");

categorias.add("Cuidado personal");

}

public boolean agregarProducto(Producto producto) {

return inventarioDAO.agregarProducto(producto);

}

public boolean eliminarProducto(String id) {

return inventarioDAO.eliminarProducto(id);

}

public boolean actualizarProducto(Producto producto) {

return inventarioDAO.actualizarProducto(producto);

}

public List<Producto> buscarPorNombre(String nombre) {

return inventarioDAO.buscarPorNombre(nombre);

}

public List<Producto> buscarPorCategoria(String categoria) {

if (categoria.equalsIgnoreCase("TODOS")) {

return inventarioDAO.obtenerTodosProductos();

}

return inventarioDAO.buscarPorCategoria(categoria);

}

public List<Producto> buscarPorProveedor(String proveedor) {

return inventarioDAO.buscarPorProveedor(proveedor);

}

public List<Producto> buscarPorEstado(String estado) {

return inventarioDAO.buscarPorEstado(estado);

}

public List<Producto> buscarPorRangoPrecio(double min, double max) {

return inventarioDAO.buscarPorRangoPrecio(min, max);

}

public List<Producto> buscarPorRangoStock(int min, int max) {

return inventarioDAO.buscarPorRangoStock(min, max);

}

public List<Producto> buscarProductosConIVA() {

return inventarioDAO.buscarProductosConIVA();

}

public List<Producto> buscarProductosSinIVA() {

return inventarioDAO.buscarProductosSinIVA();

}

public List<Producto> buscarProductosNecesitanReposicion() {

return inventarioDAO.buscarProductosNecesitanReposicion();

}

public List<Producto> buscarProductosConExcesoStock() {

return inventarioDAO.buscarProductosConExcesoStock();

}

public List<Producto> getProductos() {

return inventarioDAO.obtenerTodosProductos();

}

public List<String> getCategorias() {

List<String> categorias = new ArrayList<>();

String sql = "SELECT DISTINCT categoria FROM Productos";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

categorias.add(rs.getString("categoria"));

}

} catch (SQLException e) {

System.err.println("Error al obtener categorías: " + e.getMessage());

}

// Ordenar alfabéticamente

Collections.sort(categorias);

return categorias;

}

public Producto getProductoPorId(String id) {

return inventarioDAO.obtenerProductoPorId(id);

}

public List<Producto> getProductosBajoStock() {

return inventarioDAO.buscarProductosNecesitanReposicion();

}

public List<Producto> getProductosSobreStock() {

return inventarioDAO.buscarProductosConExcesoStock();

}

public List<Producto> buscarPorFecha(Date desde, Date hasta) {

return inventarioDAO.buscarPorFecha(desde, hasta);

}

public List<Producto> buscarPorDescripcion(String descripcion) {

return inventarioDAO.buscarPorDescripcion(descripcion);

}

}

package Controlador;

import Modelo.Devolucion;

import Modelo.DevolucionDAO;

import Modelo.Inventario;

import Modelo.InventarioDAO;

import Modelo.Producto;

import Vista.inventario;

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import ConexionBD.ConexionAccess;

public class ControladorInventario {

private Inventario modelo;

private inventario vista;

private InventarioDAO inventarioDAO;

private DevolucionDAO devolucionDAO;

public ControladorInventario(inventario vista) {

this.modelo = new Inventario();

this.inventarioDAO = new InventarioDAO();

this.devolucionDAO = new DevolucionDAO();

this.vista = vista;

if (vista != null) { // Verificar si la vista no es null

vista.setControlador(this);

}

verificarTablaDevoluciones();

}

// Métodos para manejar productos

public boolean agregarProducto(Producto producto) {

return modelo.agregarProducto(producto);

}

public boolean eliminarProducto(String id) {

return modelo.eliminarProducto(id);

}

public boolean actualizarProducto(Producto producto) {

return modelo.actualizarProducto(producto);

}

// Métodos de búsqueda

public List<Producto> buscarPorNombre(String nombre) {

return modelo.buscarPorNombre(nombre);

}

public List<Producto> buscarPorCategoria(String categoria) {

return modelo.buscarPorCategoria(categoria);

}

public List<String> getCategorias() {

// Verifica que realmente esté consultando la base de datos

System.out.println("Consultando categorías desde BD..."); // Debug

List<String> categorias = inventarioDAO.obtenerTodasCategorias();

System.out.println("Categorías obtenidas: " + categorias); // Debug

return categorias;

}

public boolean agregarCategoria(String nombre, String descripcion) {

return inventarioDAO.agregarCategoria(nombre, descripcion);

}

public boolean eliminarCategoria(String nombre) {

return inventarioDAO.eliminarCategoria(nombre);

}

public List<Producto> buscarPorProveedor(String proveedor) {

return modelo.buscarPorProveedor(proveedor);

}

public List<Producto> buscarPorEstado(String estado) {

return modelo.buscarPorEstado(estado);

}

public List<Producto> buscarPorRangoPrecio(double min, double max) {

return modelo.buscarPorRangoPrecio(min, max);

}

public List<Producto> buscarPorRangoStock(int min, int max) {

return modelo.buscarPorRangoStock(min, max);

}

public List<Producto> buscarPorFecha(Date desde, Date hasta) {

if (desde == null || hasta == null) {

throw new IllegalArgumentException("Las fechas no pueden ser nulas");

}

return modelo.buscarPorFecha(desde, hasta);

}

public List<Producto> buscarProductosConIVA() {

return modelo.buscarProductosConIVA();

}

public List<Producto> buscarProductosSinIVA() {

return modelo.buscarProductosSinIVA();

}

public List<Producto> buscarProductosNecesitanReposicion() {

return modelo.buscarProductosNecesitanReposicion();

}

public List<Producto> buscarProductosConExcesoStock() {

return modelo.buscarProductosConExcesoStock();

}

public void mostrarProductosDeCategoria(String categoria) {

vista.mostrarProductos(modelo.buscarPorCategoria(categoria));

}

public void mostrarDetalleProducto(String id) {

Producto producto = modelo.getProductoPorId(id);

if (producto != null) {

vista.mostrarDetalleProducto(producto);

}

}

public List<Producto> getTodosProductos() {

return modelo.getProductos();

}

public List<Producto> getProductosBajoStock() {

return modelo.getProductosBajoStock();

}

public List<Producto> getProductosSobreStock() {

return modelo.getProductosSobreStock();

}

public Producto getProductoPorId(String id) {

return modelo.getProductoPorId(id);

}

public List<Producto> buscarPorDescripcion(String descripcion) {

return modelo.buscarPorDescripcion(descripcion);

}

public List<Producto> buscarProductosMultiCriterio(String texto) {

List<Producto> resultados = new ArrayList<>();

// 1. Buscar por ID exacto

Producto producto = inventarioDAO.obtenerProductoPorId(texto);

if (producto != null) {

resultados.add(producto);

return resultados; // Si encontró por ID, no busca más

}

// 2. Buscar por nombre

resultados = inventarioDAO.buscarPorNombre(texto);

// 3. Si no hay resultados, buscar por descripción

if (resultados.isEmpty()) {

resultados = inventarioDAO.buscarPorDescripcion(texto);

}

// 4. Si no hay resultados, buscar por categoría

if (resultados.isEmpty()) {

resultados = inventarioDAO.buscarPorCategoria(texto);

}

return resultados;

}

private void verificarTablaDevoluciones() {

try (Connection conn = ConexionAccess.conectar()) {

DatabaseMetaData meta = conn.getMetaData();

ResultSet tables = meta.getTables(null, null, "Devoluciones", null);

if (!tables.next()) {

devolucionDAO.crearTablaDevoluciones();

}

} catch (SQLException e) {

System.err.println("Error al verificar tabla Devoluciones: " + e.getMessage());

}

}

public boolean registrarDevolucion(Devolucion devolucion) {

boolean exito = devolucionDAO.registrarDevolucion(devolucion);

if (exito) {

// Actualizar el stock del producto según el tipo de devolución

if (devolucion.getTipo().equals("CLIENTE")) {

// Devolución de cliente: aumentar stock

inventarioDAO.actualizarStock(devolucion.getIdProducto(), devolucion.getCantidad());

} else if (devolucion.getTipo().equals("PROVEEDOR")) {

// Devolución a proveedor: disminuir stock

inventarioDAO.actualizarStock(devolucion.getIdProducto(), -devolucion.getCantidad());

}

}

return exito;

}

public List<Devolucion> obtenerDevoluciones() {

return devolucionDAO.obtenerTodasDevoluciones();

}

public boolean actualizarEstadoDevolucion(String id, String nuevoEstado) {

return devolucionDAO.actualizarEstadoDevolucion(id, nuevoEstado);

}

}

package Modelo;

import ConexionBD.ConexionAccess;

import java.awt.Component;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class InventarioDAO {

public List<Producto> obtenerTodosProductos() {

List<Producto> productos = new ArrayList<>();

// Filtrar solo productos con stock disponible

String sql = "SELECT \* FROM Productos WHERE cantidad\_disponible > 0";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

Producto producto = mapearProducto(rs);

productos.add(producto);

}

} catch (SQLException e) {

System.err.println("Error al obtener productos: " + e.getMessage());

}

return productos;

}

public boolean agregarProducto(Producto producto) {

String sql = "INSERT INTO Productos (id, nombre, descripcion, categoria, proveedor, cantidad\_disponible, " +

"stock\_minimo, stock\_maximo, precio\_compra, precio\_venta, tiene\_iva, descuento, " +

"fecha\_ingreso, estado, imagen\_path, unidad\_medida) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"; // 16 parámetros

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, producto.getId());

pstmt.setString(2, producto.getNombre());

pstmt.setString(3, producto.getDescripcion());

pstmt.setString(4, producto.getCategoria());

pstmt.setString(5, producto.getProveedor());

pstmt.setInt(6, producto.getCantidadDisponible());

pstmt.setInt(7, producto.getStockMinimo());

pstmt.setInt(8, producto.getStockMaximo());

pstmt.setDouble(9, producto.getPrecioCompra());

pstmt.setDouble(10, producto.getPrecioVenta());

pstmt.setBoolean(11, producto.isTieneIVA());

pstmt.setDouble(12, producto.getDescuento());

// Manejo de fecha

if (producto.getFechaIngreso() != null) {

pstmt.setDate(13, new java.sql.Date(producto.getFechaIngreso().getTime()));

} else {

pstmt.setNull(13, java.sql.Types.DATE);

}

pstmt.setString(14, producto.getEstado());

pstmt.setString(15, producto.getImagenPath());

pstmt.setString(16, producto.getUnidadMedida()); // Unidad de medida

int affectedRows = pstmt.executeUpdate();

return affectedRows > 0;

} catch (SQLException e) {

System.err.println("Error al agregar producto: " + e.getMessage());

return false;

}

}

public boolean actualizarProducto(Producto producto) {

String sql = "UPDATE Productos SET nombre = ?, descripcion = ?, categoria = ?, proveedor = ?, " +

"cantidad\_disponible = ?, stock\_minimo = ?, stock\_maximo = ?, precio\_compra = ?, " +

"precio\_venta = ?, tiene\_iva = ?, descuento = ?, fecha\_ingreso = ?, estado = ?, " +

"imagen\_path = ?, unidad\_medida = ? WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, producto.getNombre());

pstmt.setString(2, producto.getDescripcion());

pstmt.setString(3, producto.getCategoria());

pstmt.setString(4, producto.getProveedor());

pstmt.setInt(5, producto.getCantidadDisponible());

pstmt.setInt(6, producto.getStockMinimo());

pstmt.setInt(7, producto.getStockMaximo());

pstmt.setDouble(8, producto.getPrecioCompra());

pstmt.setDouble(9, producto.getPrecioVenta());

pstmt.setBoolean(10, producto.isTieneIVA());

pstmt.setDouble(11, producto.getDescuento());

// Manejo de fecha

if (producto.getFechaIngreso() != null) {

pstmt.setDate(12, new java.sql.Date(producto.getFechaIngreso().getTime()));

} else {

pstmt.setNull(12, java.sql.Types.DATE);

}

pstmt.setString(13, producto.getEstado());

pstmt.setString(14, producto.getImagenPath());

pstmt.setString(15, producto.getUnidadMedida()); // Unidad de medida

pstmt.setString(16, producto.getId()); // ID para WHERE

int affectedRows = pstmt.executeUpdate();

return affectedRows > 0;

} catch (SQLException e) {

System.err.println("Error al actualizar producto:");

System.err.println("SQL: " + sql);

System.err.println("Producto ID: " + producto.getId());

e.printStackTrace();

return false;

}

}

public boolean eliminarProducto(String id) {

String sql = "DELETE FROM Productos WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, id);

int affectedRows = pstmt.executeUpdate();

return affectedRows > 0;

} catch (SQLException e) {

System.err.println("Error al eliminar producto: " + e.getMessage());

return false;

}

}

public List<Producto> buscarPorNombre(String nombre) {

List<Producto> productos = new ArrayList<>();

// Modificar la consulta para incluir condición de stock

String sql = "SELECT \* FROM Productos WHERE nombre LIKE ? AND cantidad\_disponible > 0";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, "%" + nombre + "%");

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por nombre: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarPorCategoria(String categoria) {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE categoria = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, categoria);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por categoría: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarPorProveedor(String proveedor) {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE proveedor LIKE ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, "%" + proveedor + "%");

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por proveedor: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarPorEstado(String estado) {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE estado = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, estado);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por estado: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarPorRangoPrecio(double min, double max) {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE precio\_venta BETWEEN ? AND ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setDouble(1, min);

pstmt.setDouble(2, max);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por rango de precio: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarPorRangoStock(int min, int max) {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE cantidad\_disponible BETWEEN ? AND ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, min);

pstmt.setInt(2, max);

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por rango de stock: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarProductosConIVA() {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE tiene\_iva = true";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar productos con IVA: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarProductosSinIVA() {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE tiene\_iva = false";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar productos sin IVA: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarProductosNecesitanReposicion() {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE cantidad\_disponible <= stock\_minimo";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar productos que necesitan reposición: " + e.getMessage());

}

return productos;

}

public List<Producto> buscarProductosConExcesoStock() {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE cantidad\_disponible > stock\_maximo";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar productos con exceso de stock: " + e.getMessage());

}

return productos;

}

public Producto obtenerProductoPorId(String id) {

String sql = "SELECT \* FROM Productos WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, id);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return mapearProducto(rs);

}

} catch (SQLException e) {

System.err.println("Error al obtener producto por ID: " + e.getMessage());

}

return null;

}

private Producto mapearProducto(ResultSet rs) throws SQLException {

return new Producto(

rs.getString("id"),

rs.getString("nombre"),

rs.getString("descripcion"),

rs.getString("categoria"),

rs.getString("proveedor"),

rs.getInt("cantidad\_disponible"),

rs.getInt("stock\_minimo"),

rs.getInt("stock\_maximo"),

rs.getDouble("precio\_compra"),

rs.getDouble("precio\_venta"),

rs.getBoolean("tiene\_iva"),

rs.getDouble("descuento"),

rs.getDate("fecha\_ingreso"),

rs.getString("estado"),

rs.getString("imagen\_path"),

rs.getString("unidad\_medida")

);

}

private void setProductoParameters(PreparedStatement pstmt, Producto producto) throws SQLException {

pstmt.setString(1, producto.getId()); // <- este es el que probablemente faltaba

pstmt.setString(2, producto.getNombre());

pstmt.setString(3, producto.getDescripcion());

pstmt.setString(4, producto.getCategoria());

pstmt.setString(5, producto.getProveedor());

pstmt.setInt(6, producto.getCantidadDisponible());

pstmt.setInt(7, producto.getStockMinimo());

pstmt.setInt(8, producto.getStockMaximo());

pstmt.setDouble(9, producto.getPrecioCompra());

pstmt.setDouble(10, producto.getPrecioVenta());

pstmt.setBoolean(11, producto.isTieneIVA());

pstmt.setDouble(12, producto.getDescuento());

pstmt.setDate(13, new java.sql.Date(producto.getFechaIngreso().getTime()));

pstmt.setString(14, producto.getEstado());

pstmt.setString(15, producto.getImagenPath());

pstmt.setString(16, producto.getUnidadMedida());

}

public List<Producto> buscarPorFecha(java.util.Date desde, java.util.Date hasta) {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE fecha\_ingreso BETWEEN ? AND ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setDate(1, new java.sql.Date(desde.getTime()));

pstmt.setDate(2, new java.sql.Date(hasta.getTime()));

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por fecha: " + e.getMessage());

}

return productos;

}

// Método para obtener todas las categorías

public List<String> obtenerTodasCategorias() {

List<String> categorias = new ArrayList<>();

try {

// Primero verifica si la tabla existe

if (!existeTabla("Categorias")) {

crearTablaCategorias(); // Método para crear la tabla si no existe

}

String sql = "SELECT nombre FROM Categorias ORDER BY nombre";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

categorias.add(rs.getString("nombre"));

}

}

} catch (SQLException e) {

System.err.println("Error crítico al obtener categorías: " + e.getMessage());

// Categorías por defecto

categorias.add("General");

categorias.add("Alimentos");

categorias.add("Bebidas");

}

return categorias;

}

private boolean existeTabla(String nombreTabla) throws SQLException {

try (Connection conn = ConexionAccess.conectar()) {

DatabaseMetaData meta = conn.getMetaData();

ResultSet tables = meta.getTables(null, null, nombreTabla, null);

return tables.next();

}

}

private void crearTablaCategorias() {

String sql = "CREATE TABLE Categorias (" +

"id AUTOINCREMENT PRIMARY KEY, " +

"nombre VARCHAR(100) NOT NULL, " +

"descripcion VARCHAR(255))";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement()) {

stmt.executeUpdate(sql);

// Insertar categorías básicas

insertarCategoriaInicial("General", "Productos generales");

insertarCategoriaInicial("Alimentos", "Productos alimenticios");

} catch (SQLException e) {

System.err.println("Error al crear tabla Categorias: " + e.getMessage());

}

}

private void insertarCategoriaInicial(String nombre, String descripcion) {

String sql = "INSERT INTO Categorias (nombre, descripcion) VALUES (?, ?)";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, nombre);

pstmt.setString(2, descripcion);

pstmt.executeUpdate();

} catch (SQLException e) {

System.err.println("Error al insertar categoría inicial: " + e.getMessage());

}

}

// Método para agregar nueva categoría

public boolean agregarCategoria(String nombre, String descripcion) {

String sql = "INSERT INTO Categorias (nombre, descripcion) VALUES (?, ?)";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, nombre);

pstmt.setString(2, descripcion);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

System.err.println("Error al agregar categoría: " + e.getMessage());

return false;

}

}

// Método para eliminar categoría

public boolean eliminarCategoria(String nombre) {

String sql = "DELETE FROM Categorias WHERE nombre = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, nombre);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

System.err.println("Error al eliminar categoría: " + e.getMessage());

return false;

}

}

public List<Producto> buscarPorDescripcion(String descripcion) {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE descripcion LIKE ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, "%" + descripcion + "%");

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

productos.add(mapearProducto(rs));

}

} catch (SQLException e) {

System.err.println("Error al buscar por descripción: " + e.getMessage());

}

return productos;

}

public Producto obtenerProductoPorNombre(String nombre) {

String sql = "SELECT \* FROM Productos WHERE nombre = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, nombre);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return mapearProducto(rs);

}

} catch (SQLException e) {

System.err.println("Error al buscar producto por nombre: " + e.getMessage());

}

return null;

}

public int obtenerIdProductoPorNombre(String nombre) {

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

conn = ConexionAccess.conectar();

String sql = "SELECT id FROM Productos WHERE nombre = ?";

pstmt = conn.prepareStatement(sql);

pstmt.setString(1, nombre);

rs = pstmt.executeQuery();

if (rs.next()) {

return rs.getInt("id");

}

return -1; // Retorna -1 si no encuentra el producto

} catch (SQLException e) {

e.printStackTrace();

return -1;

} finally {

// Cerrar recursos (rs, pstmt, conn)

}

}

public boolean actualizarStock(String idProducto, int cantidad) {

String sql = "UPDATE Productos SET cantidad\_disponible = cantidad\_disponible + ? WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, cantidad);

pstmt.setString(2, idProducto);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

System.err.println("Error al actualizar stock: " + e.getMessage());

return false;

}

}

public int obtenerStockProducto(int idProducto) throws SQLException {

String sql = "SELECT cantidad\_disponible FROM Productos WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, idProducto);

try (ResultSet rs = pstmt.executeQuery()) {

if (rs.next()) {

return rs.getInt("cantidad\_disponible");

}

}

}

throw new SQLException("Producto no encontrado");

}

public Producto obtenerProductoConBloqueo(String nombre) throws SQLException {

// Consulta modificada para Access (no soporta WITH (ROWLOCK, UPDLOCK))

String sql = "SELECT \* FROM Productos WHERE nombre = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, nombre);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return mapearProducto(rs);

}

}

return null;

}

public boolean reservarProducto(String idProducto, int cantidad) throws SQLException {

String sql = "UPDATE Productos SET cantidad\_disponible = cantidad\_disponible - ? " +

"WHERE id = ? AND cantidad\_disponible >= ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, cantidad);

pstmt.setString(2, idProducto);

pstmt.setInt(3, cantidad);

return pstmt.executeUpdate() > 0;

}

}

public List<Producto> buscarProductosSinStock() {

List<Producto> productos = new ArrayList<>();

String sql = "SELECT \* FROM Productos WHERE cantidad\_disponible = 0";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

Producto producto = mapearProducto(rs);

productos.add(producto);

}

} catch (SQLException e) {

System.err.println("Error al buscar productos sin stock: " + e.getMessage());

}

return productos;

}

}

package Modelo;

import java.util.Date;

public class Devolucion {

private String id;

private String idProducto;

private String nombreProducto;

private int cantidad;

private String tipo; // "CLIENTE" o "PROVEEDOR"

private String motivo;

private Date fecha;

private String estado; // "PENDIENTE", "PROCESADA", "RECHAZADA"

private String observaciones;

private String idTransaccionOriginal; // ID de la venta o compra original

private String idUsuario; // Usuario que registró la devolución

public Devolucion(String id, String idProducto, String nombreProducto, int cantidad,

String tipo, String motivo, Date fecha, String estado,

String observaciones, String idTransaccionOriginal, String idUsuario) {

this.id = id;

this.idProducto = idProducto;

this.nombreProducto = nombreProducto;

this.cantidad = cantidad;

this.tipo = tipo;

this.motivo = motivo;

this.fecha = fecha;

this.estado = estado;

this.observaciones = observaciones;

this.idTransaccionOriginal = idTransaccionOriginal;

this.idUsuario = idUsuario;

}

// Getters y Setters

public String getId() { return id; }

public void setId(String id) { this.id = id; }

public String getIdProducto() { return idProducto; }

public void setIdProducto(String idProducto) { this.idProducto = idProducto; }

public String getNombreProducto() { return nombreProducto; }

public void setNombreProducto(String nombreProducto) { this.nombreProducto = nombreProducto; }

public int getCantidad() { return cantidad; }

public void setCantidad(int cantidad) { this.cantidad = cantidad; }

public String getTipo() { return tipo; }

public void setTipo(String tipo) { this.tipo = tipo; }

public String getMotivo() { return motivo; }

public void setMotivo(String motivo) { this.motivo = motivo; }

public Date getFecha() { return fecha; }

public void setFecha(Date fecha) { this.fecha = fecha; }

public String getEstado() { return estado; }

public void setEstado(String estado) { this.estado = estado; }

public String getObservaciones() { return observaciones; }

public void setObservaciones(String observaciones) { this.observaciones = observaciones; }

public String getIdTransaccionOriginal() { return idTransaccionOriginal; }

public void setIdTransaccionOriginal(String idTransaccionOriginal) { this.idTransaccionOriginal = idTransaccionOriginal; }

public String getIdUsuario() { return idUsuario; }

public void setIdUsuario(String idUsuario) { this.idUsuario = idUsuario; }

}

package Modelo;

import ConexionBD.ConexionAccess;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class DevolucionDAO {

public boolean registrarDevolucion(Devolucion devolucion) {

String sql = "INSERT INTO Devoluciones (id, id\_producto, nombre\_producto, cantidad, tipo, motivo, " +

"fecha, estado, observaciones, id\_transaccion\_original, id\_usuario) " +

"VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, devolucion.getId());

pstmt.setString(2, devolucion.getIdProducto());

pstmt.setString(3, devolucion.getNombreProducto());

pstmt.setInt(4, devolucion.getCantidad());

pstmt.setString(5, devolucion.getTipo());

pstmt.setString(6, devolucion.getMotivo());

pstmt.setDate(7, new java.sql.Date(devolucion.getFecha().getTime()));

pstmt.setString(8, devolucion.getEstado());

pstmt.setString(9, devolucion.getObservaciones());

pstmt.setString(10, devolucion.getIdTransaccionOriginal());

pstmt.setString(11, devolucion.getIdUsuario());

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

System.err.println("Error al registrar devolución: " + e.getMessage());

return false;

}

}

public List<Devolucion> obtenerTodasDevoluciones() {

List<Devolucion> devoluciones = new ArrayList<>();

String sql = "SELECT \* FROM Devoluciones ORDER BY fecha DESC";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

devoluciones.add(mapearDevolucion(rs));

}

} catch (SQLException e) {

System.err.println("Error al obtener devoluciones: " + e.getMessage());

}

return devoluciones;

}

public boolean actualizarEstadoDevolucion(String id, String nuevoEstado) {

String sql = "UPDATE Devoluciones SET estado = ? WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, nuevoEstado);

pstmt.setString(2, id);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

System.err.println("Error al actualizar estado de devolución: " + e.getMessage());

return false;

}

}

private Devolucion mapearDevolucion(ResultSet rs) throws SQLException {

return new Devolucion(

rs.getString("id"),

rs.getString("id\_producto"),

rs.getString("nombre\_producto"),

rs.getInt("cantidad"),

rs.getString("tipo"),

rs.getString("motivo"),

rs.getDate("fecha"),

rs.getString("estado"),

rs.getString("observaciones"),

rs.getString("id\_transaccion\_original"),

rs.getString("id\_usuario")

);

}

public boolean crearTablaDevoluciones() {

String sql = "CREATE TABLE Devoluciones (" +

"id VARCHAR(50) PRIMARY KEY, " +

"id\_producto VARCHAR(50) NOT NULL, " +

"nombre\_producto VARCHAR(100) NOT NULL, " +

"cantidad INTEGER NOT NULL, " +

"tipo VARCHAR(20) NOT NULL, " +

"motivo VARCHAR(100) NOT NULL, " +

"fecha DATETIME NOT NULL, " +

"estado VARCHAR(20) NOT NULL, " +

"observaciones VARCHAR(255), " +

"id\_transaccion\_original VARCHAR(50), " +

"id\_usuario VARCHAR(50) NOT NULL)";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement()) {

stmt.executeUpdate(sql);

return true;

} catch (SQLException e) {

System.err.println("Error al crear tabla Devoluciones: " + e.getMessage());

return false;

}

}

}

package Vista;

import Controlador.ControladorInventario;

import Modelo.Producto;

import Modelo.Proveedor;

import Modelo.ProveedorDAO;

import Modelo.Usuario;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.nio.file.Files;

import java.nio.file.StandardCopyOption;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.sql.Timestamp;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.Date;

import javax.swing.filechooser.FileNameExtensionFilter;

import ConexionBD.ConexionAccess;

import java.util.List;

public class agregarinventario extends JDialog {

private ControladorInventario controlador;

private Usuario usuario;

private ProveedorDAO proveedorDAO;

private ActionListener guardarListener;

private boolean esEdicion = false;

private JTextField txtNombre, txtDescripcion;

private JComboBox<String> cbProveedor, cbCategoria, cbEstado;

private JSpinner spnStockMin, spnStockMax, spnCantidad;

private JSpinner spnPrecioCompra, spnPrecioVenta, spnDescuento;

private JCheckBox chkIVA;

private JButton btnSeleccionarImagen, btnAgregarProveedor;

private JLabel lblImagen;

private String imagenPath = "";

private JComboBox<String> cbUnidadMedida;

public agregarinventario(Usuario usuario, JFrame parent) {

super(parent, "Agregar Producto", true);

this.usuario = usuario;

this.proveedorDAO = new ProveedorDAO();

this.controlador = new ControladorInventario(null);

initUI();

}

public void setGuardarListener(ActionListener listener) {

this.guardarListener = listener;

this.esEdicion = true; // Indicar que estamos en modo edición

}

private void initUI() {

setSize(800, 700);

setLocationRelativeTo(getParent());

setResizable(false);

JPanel mainPanel = new JPanel(new BorderLayout(10, 10));

mainPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// Panel de título

JPanel titlePanel = new JPanel();

titlePanel.setBackground(new Color(255, 198, 144)); // Naranja claro

JLabel titleLabel = new JLabel("AGREGAR NUEVO PRODUCTO");

titleLabel.setFont(new Font("Arial", Font.BOLD, 18));

titlePanel.add(titleLabel);

mainPanel.add(titlePanel, BorderLayout.NORTH);

// Panel central con pestañas

JTabbedPane tabbedPane = new JTabbedPane();

// Pestaña 1: Información Básica

JPanel basicPanel = createBasicInfoPanel();

tabbedPane.addTab("Información Básica", basicPanel);

// Pestaña 2: Stock y Precios

JPanel stockPanel = createStockPricePanel();

tabbedPane.addTab("Stock y Precios", stockPanel);

mainPanel.add(tabbedPane, BorderLayout.CENTER);

// Panel de botones

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 10));

JButton btnGuardar = new JButton("Guardar");

JButton btnCancelar = new JButton("Cancelar");

btnGuardar.addActionListener(e -> guardarProducto());

btnCancelar.addActionListener(e -> dispose());

buttonPanel.add(btnGuardar);

buttonPanel.add(btnCancelar);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

getContentPane().add(mainPanel);

}

private JPanel createBasicInfoPanel() {

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

panel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

// Sección de imagen

JPanel imagePanel = new JPanel();

imagePanel.setBorder(BorderFactory.createTitledBorder("Imagen del Producto"));

imagePanel.setAlignmentX(Component.CENTER\_ALIGNMENT);

lblImagen = new JLabel();

lblImagen.setPreferredSize(new Dimension(200, 200));

lblImagen.setBorder(BorderFactory.createLineBorder(Color.GRAY));

btnSeleccionarImagen = new JButton("Seleccionar Imagen");

btnSeleccionarImagen.addActionListener(e -> seleccionarImagen());

imagePanel.add(lblImagen);

imagePanel.add(Box.createRigidArea(new Dimension(10, 0)));

imagePanel.add(btnSeleccionarImagen);

panel.add(imagePanel);

panel.add(Box.createRigidArea(new Dimension(0, 20)));

// Sección de datos básicos

JPanel basicInfoPanel = new JPanel();

basicInfoPanel.setLayout(new GridLayout(0, 2, 10, 10));

basicInfoPanel.setBorder(BorderFactory.createTitledBorder("Datos Básicos"));

basicInfoPanel.setBackground(new Color(230, 230, 230)); // Gris claro

// Nombre

basicInfoPanel.add(new JLabel("Nombre:"));

txtNombre = new JTextField();

basicInfoPanel.add(txtNombre);

// Descripción

basicInfoPanel.add(new JLabel("Descripción:"));

txtDescripcion = new JTextField();

basicInfoPanel.add(txtDescripcion);

// Categoría

basicInfoPanel.add(new JLabel("Categoría:"));

cbCategoria = new JComboBox<>(controlador.getCategorias().toArray(new String[0]));

cbCategoria.removeItem("TODOS");

basicInfoPanel.add(cbCategoria);

// Proveedor

basicInfoPanel.add(new JLabel("Proveedor:"));

JPanel proveedorPanel = new JPanel(new BorderLayout());

cbProveedor = new JComboBox<>(cargarProveedores());

btnAgregarProveedor = new JButton("+");

btnAgregarProveedor.addActionListener(e -> agregarNuevoProveedor());

proveedorPanel.add(cbProveedor, BorderLayout.CENTER);

proveedorPanel.add(btnAgregarProveedor, BorderLayout.EAST);

basicInfoPanel.add(proveedorPanel);

// Estado

basicInfoPanel.add(new JLabel("Estado:"));

cbEstado = new JComboBox<>(new String[]{"Activo", "Descontinuado", "Dañado"});

basicInfoPanel.add(cbEstado);

basicInfoPanel.add(new JLabel("Unidad de Medida:"));

String[] unidades = {"Pieza", "Kg", "Litro", "Metro", "Caja", "Paquete"};

cbUnidadMedida = new JComboBox<>(unidades); // Usar la variable de instancia

basicInfoPanel.add(cbUnidadMedida);

panel.add(basicInfoPanel);

return panel;

}

private JPanel createStockPricePanel() {

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

panel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

// Sección de stock

JPanel stockPanel = new JPanel(new GridLayout(0, 2, 10, 10));

stockPanel.setBorder(BorderFactory.createTitledBorder("Stock"));

stockPanel.setBackground(new Color(255, 182, 193)); // Rosa

stockPanel.add(new JLabel("Cantidad disponible:"));

spnCantidad = new JSpinner(new SpinnerNumberModel(0, 0, 10000, 1));

stockPanel.add(spnCantidad);

stockPanel.add(new JLabel("Stock mínimo:"));

spnStockMin = new JSpinner(new SpinnerNumberModel(0, 0, 1000, 1));

stockPanel.add(spnStockMin);

stockPanel.add(new JLabel("Stock máximo:"));

spnStockMax = new JSpinner(new SpinnerNumberModel(0, 0, 10000, 1));

stockPanel.add(spnStockMax);

panel.add(stockPanel);

panel.add(Box.createRigidArea(new Dimension(0, 20)));

// Sección financiera

JPanel financePanel = new JPanel(new GridLayout(0, 2, 10, 10));

financePanel.setBorder(BorderFactory.createTitledBorder("Datos Financieros"));

financePanel.setBackground(new Color(216, 237, 88)); // Amarillo claro

// Precio compra (formato moneda)

financePanel.add(new JLabel("Precio compra:"));

spnPrecioCompra = new JSpinner(new SpinnerNumberModel(0.0, 0.0, 100000.0, 0.5));

JSpinner.NumberEditor editorCompra = new JSpinner.NumberEditor(spnPrecioCompra, "$#,##0.00");

spnPrecioCompra.setEditor(editorCompra);

financePanel.add(spnPrecioCompra);

// Precio venta (formato moneda)

financePanel.add(new JLabel("Precio venta:"));

spnPrecioVenta = new JSpinner(new SpinnerNumberModel(0.0, 0.0, 100000.0, 0.5));

JSpinner.NumberEditor editorVenta = new JSpinner.NumberEditor(spnPrecioVenta, "$#,##0.00");

spnPrecioVenta.setEditor(editorVenta);

financePanel.add(spnPrecioVenta);

// IVA

financePanel.add(new JLabel("IVA:"));

chkIVA = new JCheckBox("Aplica IVA (16%)");

financePanel.add(chkIVA);

// Descuento

financePanel.add(new JLabel("Descuento (%):"));

spnDescuento = new JSpinner(new SpinnerNumberModel(0.0, 0.0, 100.0, 0.5));

financePanel.add(spnDescuento);

panel.add(financePanel);

return panel;

}

private String[] cargarProveedores() {

return proveedorDAO.obtenerTodosProveedores().stream()

.map(Proveedor::getNombre)

.toArray(String[]::new);

}

private void seleccionarImagen() {

JFileChooser fileChooser = new JFileChooser();

fileChooser.setDialogTitle("Seleccionar imagen del producto");

fileChooser.setAcceptAllFileFilterUsed(false);

fileChooser.addChoosableFileFilter(

new FileNameExtensionFilter("Imágenes", "jpg", "jpeg", "png", "gif"));

int returnValue = fileChooser.showOpenDialog(this);

if (returnValue == JFileChooser.APPROVE\_OPTION) {

File selectedFile = fileChooser.getSelectedFile();

try {

// Crear directorio de imágenes si no existe

File imgDir = new File("imagenes\_productos");

if (!imgDir.exists()) {

imgDir.mkdir();

}

// Copiar imagen al directorio de la aplicación

File destino = new File(imgDir, selectedFile.getName());

Files.copy(selectedFile.toPath(), destino.toPath(), StandardCopyOption.REPLACE\_EXISTING);

// Mostrar imagen en el label

ImageIcon icon = new ImageIcon(destino.getAbsolutePath());

Image img = icon.getImage().getScaledInstance(

lblImagen.getWidth(), lblImagen.getHeight(), Image.SCALE\_SMOOTH);

lblImagen.setIcon(new ImageIcon(img));

// Guardar ruta para la base de datos

imagenPath = destino.getAbsolutePath();

} catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Error al cargar la imagen: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

private void agregarNuevoProveedor() {

// Crear diálogo para agregar proveedor rápido

JDialog dialog = new JDialog(this, "Agregar Proveedor Rápido", true);

dialog.setSize(400, 300);

dialog.setLocationRelativeTo(this);

JPanel panel = new JPanel(new GridLayout(0, 2, 10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

JTextField txtNombre = new JTextField();

JTextField txtTelefono = new JTextField();

panel.add(new JLabel("Nombre:"));

panel.add(txtNombre);

panel.add(new JLabel("Teléfono:"));

panel.add(txtTelefono);

JButton btnGuardar = new JButton("Guardar");

JButton btnCancelar = new JButton("Cancelar");

btnGuardar.addActionListener(e -> {

if (txtNombre.getText().trim().isEmpty() || txtTelefono.getText().trim().isEmpty()) {

JOptionPane.showMessageDialog(dialog,

"Nombre y teléfono son obligatorios", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

Proveedor nuevo = new Proveedor(

"PRV-" + System.currentTimeMillis(),

txtNombre.getText(),

txtTelefono.getText(),

"",

"",

new Timestamp(System.currentTimeMillis())

);

try {

if (proveedorDAO.agregarProveedor(nuevo)) {

cbProveedor.addItem(nuevo.getNombre());

cbProveedor.setSelectedItem(nuevo.getNombre());

dialog.dispose();

} else {

JOptionPane.showMessageDialog(dialog,

"Error al guardar proveedor", "Error", JOptionPane.ERROR\_MESSAGE);

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(dialog,

"Error al guardar proveedor: " + ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

ex.printStackTrace();

}

});

btnCancelar.addActionListener(e -> dialog.dispose());

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

buttonPanel.add(btnCancelar);

buttonPanel.add(btnGuardar);

dialog.getContentPane().add(panel, BorderLayout.CENTER);

dialog.getContentPane().add(buttonPanel, BorderLayout.SOUTH);

dialog.setVisible(true);

}

// Método para obtener los datos del formulario como Producto

public Producto obtenerProductoDelFormulario() {

// Obtener el proveedor seleccionado

String nombreProveedor = (String) cbProveedor.getSelectedItem();

Proveedor proveedor = proveedorDAO.obtenerTodosProveedores().stream()

.filter(p -> p.getNombre().equals(nombreProveedor))

.findFirst()

.orElse(null);

if (proveedor == null) {

throw new IllegalStateException("Proveedor no válido");

}

return new Producto(

generarId(), // Esto se sobrescribirá con el ID original en edición

txtNombre.getText(),

txtDescripcion.getText(),

(String) cbCategoria.getSelectedItem(),

proveedor.getId(),

(Integer) spnCantidad.getValue(),

(Integer) spnStockMin.getValue(),

(Integer) spnStockMax.getValue(),

(Double) spnPrecioCompra.getValue(),

(Double) spnPrecioVenta.getValue(),

chkIVA.isSelected(),

(Double) spnDescuento.getValue(),

new Date(), // Usar fecha actual o mantener la original?

(String) cbEstado.getSelectedItem(),

imagenPath,

(String) cbUnidadMedida.getSelectedItem() // Unidad de medida agregada

);

}

private void guardarProducto() {

if (esEdicion && guardarListener != null) {

guardarListener.actionPerformed(new ActionEvent(this, ActionEvent.ACTION\_PERFORMED, "guardar"));

return;

}

// Validaciones adicionales

if (txtNombre.getText().trim().isEmpty()) {

JOptionPane.showMessageDialog(this, "El nombre del producto es obligatorio", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

if ((Integer)spnStockMin.getValue() > (Integer)spnStockMax.getValue()) {

JOptionPane.showMessageDialog(this, "El stock mínimo no puede ser mayor al máximo", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

if ((Double)spnPrecioCompra.getValue() <= 0 || (Double)spnPrecioVenta.getValue() <= 0) {

JOptionPane.showMessageDialog(this, "Los precios deben ser mayores a cero", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Validar que el precio de venta sea mayor que el de compra

if ((Double)spnPrecioVenta.getValue() <= (Double)spnPrecioCompra.getValue()) {

JOptionPane.showMessageDialog(this,

"El precio de venta debe ser mayor al precio de compra",

"Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Validar que el stock actual esté entre mínimo y máximo

int stockActual = (Integer)spnCantidad.getValue();

int stockMin = (Integer)spnStockMin.getValue();

int stockMax = (Integer)spnStockMax.getValue();

if (stockActual < 0 || stockMin < 0 || stockMax < 0) {

JOptionPane.showMessageDialog(this,

"Los valores de stock no pueden ser negativos",

"Error", JOptionPane.ERROR\_MESSAGE);

return;

}

if (stockMin >= stockMax) {

JOptionPane.showMessageDialog(this,

"El stock mínimo debe ser menor al stock máximo",

"Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Obtener el proveedor seleccionado

String nombreProveedor = (String) cbProveedor.getSelectedItem();

Proveedor proveedor = proveedorDAO.obtenerTodosProveedores().stream()

.filter(p -> p.getNombre().equals(nombreProveedor))

.findFirst()

.orElse(null);

if (proveedor == null) {

JOptionPane.showMessageDialog(this,

"Seleccione un proveedor válido", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

// Crear el producto

Producto producto = new Producto(

generarId(), // Esto se sobrescribirá con el ID original en edición

txtNombre.getText(),

txtDescripcion.getText(),

(String) cbCategoria.getSelectedItem(),

proveedor.getId(),

(Integer) spnCantidad.getValue(),

(Integer) spnStockMin.getValue(),

(Integer) spnStockMax.getValue(),

(Double) spnPrecioCompra.getValue(),

(Double) spnPrecioVenta.getValue(),

chkIVA.isSelected(),

(Double) spnDescuento.getValue(),

new Date(), // Usar fecha actual o mantener la original?

(String) cbEstado.getSelectedItem(),

imagenPath,

(String) cbUnidadMedida.getSelectedItem() // Unidad de medida agregada

);

try {

if (controlador.agregarProducto(producto)) {

JOptionPane.showMessageDialog(this, "Producto agregado correctamente", "Éxito", JOptionPane.INFORMATION\_MESSAGE);

dispose();

} else {

JOptionPane.showMessageDialog(this, "Error al agregar producto", "Error", JOptionPane.ERROR\_MESSAGE);

}

} catch (Exception e) {

JOptionPane.showMessageDialog(this, "Error al guardar: " + e.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

e.printStackTrace();

}

}

private String generarId() {

return "PROD-" + System.currentTimeMillis() + "-" + (int)(Math.random() \* 1000);

}

public void cargarDatosProducto(Producto producto) {

// Cargar datos básicos

txtNombre.setText(producto.getNombre());

txtDescripcion.setText(producto.getDescripcion());

cbCategoria.setSelectedItem(producto.getCategoria());

// Cargar proveedor

Proveedor prov = proveedorDAO.buscarProveedorPorId(producto.getProveedor());

if (prov != null) {

cbProveedor.setSelectedItem(prov.getNombre());

}

cbEstado.setSelectedItem(producto.getEstado());

// Cargar stock

spnCantidad.setValue(producto.getCantidadDisponible());

spnStockMin.setValue(producto.getStockMinimo());

spnStockMax.setValue(producto.getStockMaximo());

// Cargar datos financieros

spnPrecioCompra.setValue(producto.getPrecioCompra());

spnPrecioVenta.setValue(producto.getPrecioVenta());

chkIVA.setSelected(producto.isTieneIVA());

spnDescuento.setValue(producto.getDescuento());

// Cargar imagen si existe

if (producto.getImagenPath() != null && !producto.getImagenPath().isEmpty()) {

try {

ImageIcon icon = new ImageIcon(producto.getImagenPath());

// Verificar dimensiones antes de escalar

int width = lblImagen.getWidth() > 0 ? lblImagen.getWidth() : 200;

int height = lblImagen.getHeight() > 0 ? lblImagen.getHeight() : 200;

Image img = icon.getImage().getScaledInstance(width, height, Image.SCALE\_SMOOTH);

lblImagen.setIcon(new ImageIcon(img));

imagenPath = producto.getImagenPath();

} catch (Exception e) {

System.err.println("Error al cargar imagen: " + e.getMessage());

lblImagen.setIcon(null);

}

} else {

lblImagen.setIcon(null);

}

}

public List<String> getCategorias() {

List<String> categorias = new ArrayList<>();

String sql = "SELECT DISTINCT categoria FROM Productos WHERE categoria IS NOT NULL";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

String categoria = rs.getString("categoria");

if (categoria != null && !categoria.trim().isEmpty()) {

categorias.add(categoria.trim());

}

}

// Si no hay categorías, agregar algunas básicas

if (categorias.isEmpty()) {

categorias.add("Abarrotes");

categorias.add("Bebidas");

categorias.add("Lácteos");

}

} catch (SQLException e) {

System.err.println("Error al obtener categorías: " + e.getMessage());

// Retornar categorías predeterminadas en caso de error

categorias.add("General");

}

Collections.sort(categorias);

return categorias;

}

}

package Vista;

import java.awt.BorderLayout;

import java.awt.EventQueue;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.border.EmptyBorder;

import javax.swing.table.DefaultTableModel;

import Controlador.ControladorInventario;

public class GestionCategorias extends JDialog {

private ControladorInventario controlador;

private JTable tablaCategorias;

public GestionCategorias(JFrame parent, ControladorInventario controlador) {

super(parent, "Gestión de Categorías", true);

this.controlador = controlador;

initUI();

}

private void initUI() {

setSize(500, 400);

// Panel principal

JPanel mainPanel = new JPanel(new BorderLayout());

// Tabla de categorías

String[] columnNames = {"Nombre", "Descripción"};

DefaultTableModel model = new DefaultTableModel(columnNames, 0);

tablaCategorias = new JTable(model);

actualizarTabla();

mainPanel.add(new JScrollPane(tablaCategorias), BorderLayout.CENTER);

// Panel de botones

JPanel buttonPanel = new JPanel();

JButton btnAgregar = new JButton("Agregar");

btnAgregar.addActionListener(e -> agregarCategoria());

JButton btnEliminar = new JButton("Eliminar");

btnEliminar.addActionListener(e -> eliminarCategoria());

buttonPanel.add(btnAgregar);

buttonPanel.add(btnEliminar);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

getContentPane().add(mainPanel);

}

private void actualizarTabla() {

DefaultTableModel model = (DefaultTableModel) tablaCategorias.getModel();

model.setRowCount(0);

List<String> categorias = controlador.getCategorias();

for (String categoria : categorias) {

model.addRow(new Object[]{categoria, ""}); // Puedes cargar descripción si la muestras

}

}

private void agregarCategoria() {

JTextField txtNombre = new JTextField();

JTextField txtDescripcion = new JTextField();

Object[] fields = {

"Nombre:", txtNombre,

"Descripción:", txtDescripcion

};

int option = JOptionPane.showConfirmDialog(

this,

fields,

"Nueva Categoría",

JOptionPane.OK\_CANCEL\_OPTION

);

if (option == JOptionPane.OK\_OPTION) {

if (controlador.agregarCategoria(txtNombre.getText(), txtDescripcion.getText())) {

actualizarTabla();

} else {

JOptionPane.showMessageDialog(this, "Error al agregar categoría");

}

}

}

private void eliminarCategoria() {

// Implementar lógica de eliminación

int selectedRow = tablaCategorias.getSelectedRow();

if (selectedRow != -1) {

String nombreCategoria = (String) tablaCategorias.getValueAt(selectedRow, 0);

if (controlador.eliminarCategoria(nombreCategoria)) {

((DefaultTableModel) tablaCategorias.getModel()).removeRow(selectedRow);

} else {

JOptionPane.showMessageDialog(this, "Error al eliminar categoría");

}

}

}

}

package Vista;

import Controlador.ControladorInventario;

import Controlador.ReportesControlador;

import Modelo.ClienteDAOImpl;

import Modelo.Devolucion;

import Modelo.Producto;

import Modelo.Proveedor;

import Modelo.ProveedorDAO;

import Modelo.Usuario;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import org.apache.poi.ss.usermodel.Cell;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import com.itextpdf.text.BaseColor;

import com.itextpdf.text.Document;

import com.itextpdf.text.Element;

import com.itextpdf.text.FontFactory;

import com.itextpdf.text.Paragraph;

import com.itextpdf.text.Phrase;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.FocusAdapter;

import java.awt.event.FocusEvent;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.FileOutputStream;

import java.io.FileWriter;

import java.text.NumberFormat;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

public class inventario extends JFrame {

private ControladorInventario controlador;

private Usuario usuario;

private ProveedorDAO proveedorDAO;

private JPanel panelProductos;

private JDialog detallesDialog;

private JLabel lblImagen;

private JLabel lblNombre;

private JLabel lblId;

private JLabel lblDescripcion;

private JLabel lblCategoria;

private JLabel lblProveedor;

private JLabel lblStock;

private JLabel lblStockMin;

private JLabel lblStockMax;

private JLabel lblPrecioVenta;

private JLabel lblPrecioCompra;

private JLabel lblIVA;

private JLabel lblDescuento;

private JLabel lblFecha;

private JLabel lblEstado;

private JButton btnEditar;

private JButton btnEliminar;

private JButton btnContactarProveedor;

private Producto productoSeleccionado;

private Component panel;

private JTextField txtBusqueda;

private JButton btnBusquedaAvanzada;

public inventario(Usuario usuario) {

this.usuario = usuario;

this.controlador = new ControladorInventario(this);

this.proveedorDAO = new ProveedorDAO();

initUI();

}

private void initUI() {

setTitle("El Habanerito - Inventario");

setSize(1517, 903);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

setResizable(true);

// Panel principal con BorderLayout

JPanel mainPanel = new JPanel(new BorderLayout());

// 1. Panel superior con logo y usuario (NORTH)

JPanel headerPanel = createHeaderPanel();

mainPanel.add(headerPanel, BorderLayout.NORTH);

// 2. Menú horizontal justo debajo del header (en su propio panel)

JPanel menuContainer = new JPanel(new BorderLayout());

JPanel menuPanel = crearMenuHorizontal();

menuContainer.add(menuPanel, BorderLayout.NORTH);

// 3. Panel central con SplitPane para filtros y productos

JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL\_SPLIT);

// Panel izquierdo con filtros (botones rosas)

JPanel filterPanel = createCategoryFilterPanel();

filterPanel.setPreferredSize(new Dimension(220, 0));

splitPane.setLeftComponent(filterPanel);

// Panel derecho con productos

panelProductos = new JPanel(new GridLayout(0, 3, 15, 15));

JScrollPane scrollProductos = new JScrollPane(panelProductos);

scrollProductos.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

splitPane.setRightComponent(scrollProductos);

splitPane.setDividerLocation(220);

splitPane.setResizeWeight(0);

menuContainer.add(splitPane, BorderLayout.CENTER);

// 4. Panel inferior con búsqueda y botones (SOUTH)

JPanel bottomSearchPanel = new JPanel(new BorderLayout());

bottomSearchPanel.setBackground(new Color(255, 182, 193));

bottomSearchPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Panel de búsqueda (izquierda)

JPanel searchPanel = createSearchPanel();

searchPanel.setBackground(new Color(255, 182, 193));

bottomSearchPanel.add(searchPanel, BorderLayout.CENTER);

// Panel de botones (derecha)

JPanel buttonPanel = createBottomPanel();

buttonPanel.setBackground(new Color(255, 182, 193));

bottomSearchPanel.add(buttonPanel, BorderLayout.EAST);

menuContainer.add(bottomSearchPanel, BorderLayout.SOUTH);

mainPanel.add(menuContainer, BorderLayout.CENTER);

getContentPane().add(mainPanel);

// Cargar productos iniciales

mostrarProductos(controlador.getTodosProductos());

}

private JPanel createTopPanel() {

JPanel topContainer = new JPanel();

topContainer.setLayout(new BoxLayout(topContainer, BoxLayout.Y\_AXIS));

// Panel de encabezado

JPanel headerPanel = createHeaderPanel();

topContainer.add(headerPanel);

// Menú horizontal

JPanel horizontalMenu = crearMenuHorizontal();

topContainer.add(horizontalMenu);

// Panel de búsqueda rápida

JPanel searchPanel = createSearchPanel();

topContainer.add(searchPanel);

return topContainer;

}

private JPanel createHeaderPanel() {

JPanel panel = new JPanel(new BorderLayout());

panel.setBackground(new Color(255, 198, 144));

panel.setPreferredSize(new Dimension(getWidth(), 80));

panel.setBorder(BorderFactory.createEmptyBorder(10, 20, 10, 20));

JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 0));

rightPanel.setOpaque(false);

JButton usuarioBtn = new JButton(usuario.getUsername());

usuarioBtn.setFont(new Font("Arial", Font.BOLD, 14));

usuarioBtn.setForeground(Color.BLACK);

usuarioBtn.setContentAreaFilled(false);

usuarioBtn.setBorderPainted(false);

usuarioBtn.setFocusPainted(false);

usuarioBtn.setCursor(new Cursor(Cursor.HAND\_CURSOR));

usuarioBtn.addActionListener(e -> cambiarUsuario());

rightPanel.add(usuarioBtn);

try {

ImageIcon originalIcon = new ImageIcon("imagen\\logo.png");

Image originalImage = originalIcon.getImage();

int logoHeight = 60;

int logoWidth = (int) ((double) originalIcon.getIconWidth() / originalIcon.getIconHeight() \* logoHeight);

Image resizedImage = originalImage.getScaledInstance(logoWidth, logoHeight, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage));

rightPanel.add(logo, 0);

} catch (Exception e) {

System.err.println("Error cargando el logo: " + e.getMessage());

}

panel.add(rightPanel, BorderLayout.EAST);

return panel;

}

private JPanel crearMenuHorizontal() {

JPanel menuPanel = new JPanel(new GridLayout(1, 5));

menuPanel.setBackground(new Color(230, 230, 230));

menuPanel.setPreferredSize(new Dimension(0, 50));

menuPanel.setBorder(BorderFactory.createMatteBorder(1, 0, 1, 0, Color.GRAY));

String[] opciones = {"Productos", "Reportes", "Inventario", "Cliente","Proveedores","Usuarios", "Salir"};

for (String opcion : opciones) {

JButton btn = crearBotonMenu(opcion);

btn.addActionListener(e -> manejarAccionMenu(opcion));

menuPanel.add(btn);

}

return menuPanel;

}

private JPanel createSearchPanel() {

JPanel panel = new JPanel(new BorderLayout());

panel.setBorder(BorderFactory.createEmptyBorder(15, 20, 15, 20));

panel.setBackground(new Color(240, 240, 240));

// Panel contenedor

JPanel searchContainer = new JPanel(new BorderLayout());

searchContainer.setBackground(Color.WHITE);

searchContainer.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(200, 200, 200), 1),

BorderFactory.createEmptyBorder(2, 10, 2, 2)

));

// Campo de búsqueda

txtBusqueda = new JTextField();

txtBusqueda.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

txtBusqueda.setFont(new Font("Segoe UI", Font.PLAIN, 14));

// Botón de búsqueda ovalado

JButton btnBuscar = createOvalSearchButton();

searchContainer.add(txtBusqueda, BorderLayout.CENTER);

searchContainer.add(btnBuscar, BorderLayout.EAST);

panel.add(searchContainer, BorderLayout.CENTER);

return panel;

}

private JButton createOvalSearchButton() {

JButton button = new JButton() {

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Fondo ovalado

g2.setColor(new Color(80, 80, 80));

g2.fillOval(0, 0, getWidth(), getHeight());

g2.dispose();

super.paintComponent(g);

}

};

// Cargar imagen (versión segura)

try {

ImageIcon icon = new ImageIcon(getClass().getResource("imagen\\buscar.png"));

if (icon.getImage() != null) {

// Escalar imagen

Image img = icon.getImage().getScaledInstance(20, 20, Image.SCALE\_SMOOTH);

button.setIcon(new ImageIcon(img));

}

} catch (Exception e) {

// Dibujar lupa manualmente si no hay imagen

button.setIcon(null);

}

button.setContentAreaFilled(false);

button.setBorderPainted(false);

button.setFocusPainted(false);

button.setPreferredSize(new Dimension(40, 40));

button.setCursor(new Cursor(Cursor.HAND\_CURSOR));

button.addActionListener(e -> realizarBusqueda());

// Efecto hover

button.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

button.setBackground(new Color(100, 100, 100));

}

@Override

public void mouseExited(MouseEvent e) {

button.setBackground(new Color(80, 80, 80));

}

});

return button;

}

private void realizarBusqueda() {

// Ignorar si es el texto de placeholder

if (txtBusqueda.getText().equals("Buscar productos...")) {

mostrarProductos(controlador.getTodosProductos());

return;

}

String texto = txtBusqueda.getText().trim();

if (texto.isEmpty()) {

mostrarProductos(controlador.getTodosProductos());

return;

}

// Mostrar animación de carga (opcional)

JLabel loading = new JLabel("Buscando...", SwingConstants.CENTER);

JOptionPane.showMessageDialog(this, loading, "", JOptionPane.PLAIN\_MESSAGE);

SwingWorker<List<Producto>, Void> worker = new SwingWorker<>() {

@Override

protected List<Producto> doInBackground() throws Exception {

return controlador.buscarProductosMultiCriterio(texto);

}

@Override

protected void done() {

try {

List<Producto> resultados = get();

if (resultados != null && !resultados.isEmpty()) {

mostrarProductos(resultados);

} else {

JOptionPane.showMessageDialog(inventario.this,

"No se encontraron productos",

"Búsqueda", JOptionPane.INFORMATION\_MESSAGE);

}

} catch (Exception e) {

JOptionPane.showMessageDialog(inventario.this,

"Error en la búsqueda: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

};

worker.execute();

}

private JButton createActionButton(String text) {

JButton btn = new JButton(text);

btn.setPreferredSize(new Dimension(150, 30));

btn.setFont(new Font("Arial", Font.BOLD, 12));

btn.setBackground(new Color(70, 130, 180)); // Azul

btn.setForeground(Color.WHITE);

btn.setFocusPainted(false);

btn.setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));

btn.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

btn.setBackground(new Color(50, 110, 160));

}

@Override

public void mouseExited(MouseEvent e) {

btn.setBackground(new Color(70, 130, 180));

}

});

return btn;

}

public void mostrarProductos(List<Producto> productos) {

panelProductos.removeAll();

for (Producto producto : productos) {

JPanel card = createProductCard(producto);

panelProductos.add(card);

}

panelProductos.revalidate();

panelProductos.repaint();

}

private JPanel createProductCard(Producto producto) {

// Panel principal de la tarjeta

JPanel card = new JPanel();

card.setLayout(new BoxLayout(card, BoxLayout.Y\_AXIS));

card.setBackground(Color.WHITE);

card.setPreferredSize(new Dimension(240, 300));

// Estilo del borde basado en el estado del stock

if (producto.necesitaReposicion()) {

card.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(220, 80, 80), 2),

BorderFactory.createEmptyBorder(10, 10, 10, 10)

));

} else {

card.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(220, 220, 220)),

BorderFactory.createEmptyBorder(10, 10, 10, 10)

));

}

// Panel para la imagen del producto

JPanel imagePanel = new JPanel();

imagePanel.setBackground(Color.WHITE);

imagePanel.setAlignmentX(Component.CENTER\_ALIGNMENT);

JLabel imgLabel = new JLabel();

imgLabel.setPreferredSize(new Dimension(150, 150));

imgLabel.setHorizontalAlignment(JLabel.CENTER);

// Cargar imagen del producto o usar placeholder

if (producto.getImagenPath() != null && !producto.getImagenPath().isEmpty()) {

try {

ImageIcon icon = new ImageIcon(producto.getImagenPath());

Image img = icon.getImage().getScaledInstance(150, 150, Image.SCALE\_SMOOTH);

imgLabel.setIcon(new ImageIcon(img));

} catch (Exception e) {

imgLabel.setIcon(new ImageIcon(createPlaceholderImage(producto.getNombre(), 150, 150)));

}

} else {

imgLabel.setIcon(new ImageIcon(createPlaceholderImage(producto.getNombre(), 150, 150)));

}

imagePanel.add(imgLabel);

card.add(imagePanel);

card.add(Box.createVerticalStrut(10));

// Panel para la información del producto

JPanel infoPanel = new JPanel();

infoPanel.setLayout(new BoxLayout(infoPanel, BoxLayout.Y\_AXIS));

infoPanel.setBackground(Color.WHITE);

infoPanel.setAlignmentX(Component.CENTER\_ALIGNMENT);

// Nombre del producto

JLabel nameLabel = new JLabel(producto.getNombre());

nameLabel.setFont(new Font("Segoe UI", Font.BOLD, 14));

nameLabel.setForeground(new Color(50, 50, 50));

nameLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

infoPanel.add(nameLabel);

infoPanel.add(Box.createVerticalStrut(5));

// ID del producto

JLabel idLabel = new JLabel("ID: " + producto.getId());

idLabel.setFont(new Font("Segoe UI", Font.PLAIN, 11));

idLabel.setForeground(new Color(100, 100, 100));

idLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

infoPanel.add(idLabel);

infoPanel.add(Box.createVerticalStrut(8));

// Precio y descuento

JPanel pricePanel = new JPanel();

pricePanel.setBackground(Color.WHITE);

pricePanel.setAlignmentX(Component.CENTER\_ALIGNMENT);

JLabel priceLabel = new JLabel(String.format("$%.2f", producto.getPrecioVenta()));

priceLabel.setFont(new Font("Segoe UI", Font.BOLD, 14));

priceLabel.setForeground(new Color(0, 100, 0));

// Mostrar descuento si aplica

if (producto.getDescuento() > 0) {

JLabel discountLabel = new JLabel(String.format(" (%.0f%% OFF)", producto.getDescuento()));

discountLabel.setFont(new Font("Segoe UI", Font.BOLD, 11));

discountLabel.setForeground(new Color(200, 0, 0));

pricePanel.add(discountLabel);

}

pricePanel.add(priceLabel);

infoPanel.add(pricePanel);

infoPanel.add(Box.createVerticalStrut(8));

// Stock y unidad de medida

JLabel stockLabel = new JLabel();

stockLabel.setFont(new Font("Segoe UI", Font.PLAIN, 12));

stockLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

if (producto.necesitaReposicion()) {

stockLabel.setText(String.format("¡BAJO STOCK! (%d %s)",

producto.getCantidadDisponible(),

producto.getUnidadMedida()));

stockLabel.setForeground(Color.RED);

stockLabel.setFont(stockLabel.getFont().deriveFont(Font.BOLD));

} else {

stockLabel.setText(String.format("Disponible: %d %s",

producto.getCantidadDisponible(),

producto.getUnidadMedida()));

stockLabel.setForeground(new Color(70, 70, 70));

}

infoPanel.add(stockLabel);

infoPanel.add(Box.createVerticalStrut(5));

// Estado del producto

JLabel statusLabel = new JLabel("Estado: " + producto.getEstado());

statusLabel.setFont(new Font("Segoe UI", Font.PLAIN, 11));

statusLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

// Color según estado

switch(producto.getEstado().toLowerCase()) {

case "activo":

statusLabel.setForeground(new Color(0, 120, 0));

break;

case "descontinuado":

statusLabel.setForeground(new Color(150, 150, 150));

break;

case "dañado":

statusLabel.setForeground(new Color(200, 0, 0));

break;

default:

statusLabel.setForeground(new Color(70, 70, 70));

}

infoPanel.add(statusLabel);

card.add(infoPanel);

// Evento para mostrar detalles al hacer clic

card.addMouseListener(new MouseAdapter() {

@Override

public void mouseClicked(MouseEvent e) {

mostrarDetalleProducto(producto);

}

@Override

public void mouseEntered(MouseEvent e) {

card.setBackground(new Color(245, 245, 245));

}

@Override

public void mouseExited(MouseEvent e) {

card.setBackground(Color.WHITE);

}

});

return card;

}

public void mostrarDetalleProducto(Producto producto) {

this.productoSeleccionado = producto;

// Configuración del diálogo

detallesDialog = new JDialog(this, "Detalles del Producto", true);

detallesDialog.setSize(700, 800);

detallesDialog.setLocationRelativeTo(this);

detallesDialog.setLayout(new BorderLayout());

detallesDialog.getContentPane().setBackground(new Color(240, 245, 250));

// Panel principal con margenes

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y\_AXIS));

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

mainPanel.setBackground(new Color(240, 245, 250));

// 1. Panel de imagen

JPanel imagePanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

imagePanel.setBackground(Color.WHITE);

imagePanel.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(220, 220, 230), 1),

BorderFactory.createEmptyBorder(15, 15, 15, 15)

));

lblImagen = new JLabel();

lblImagen.setPreferredSize(new Dimension(250, 250));

// Cargar imagen o placeholder

if (producto.getImagenPath() != null && !producto.getImagenPath().isEmpty()) {

try {

ImageIcon icon = new ImageIcon(producto.getImagenPath());

Image img = icon.getImage().getScaledInstance(250, 250, Image.SCALE\_SMOOTH);

lblImagen.setIcon(new ImageIcon(img));

} catch (Exception e) {

lblImagen.setIcon(new ImageIcon(createPlaceholderImage("Imagen no disponible", 250, 250)));

}

} else {

lblImagen.setIcon(new ImageIcon(createPlaceholderImage(producto.getNombre(), 250, 250)));

}

imagePanel.add(lblImagen);

mainPanel.add(imagePanel);

mainPanel.add(Box.createVerticalStrut(20));

// 2. Panel de pestañas

JTabbedPane tabbedPane = new JTabbedPane();

tabbedPane.setFont(new Font("Segoe UI", Font.PLAIN, 12));

// Pestaña de Información General

JPanel infoPanel = createInfoPanel(producto);

tabbedPane.addTab("Información General", infoPanel);

// Pestaña de Stock y Precios

JPanel stockPanel = createStockPanel(producto);

tabbedPane.addTab("Stock y Precios", stockPanel);

// Pestaña de Proveedor (si existe)

if (producto.getProveedor() != null && !producto.getProveedor().isEmpty()) {

JPanel supplierPanel = createSupplierPanel(producto);

tabbedPane.addTab("Proveedor", supplierPanel);

}

mainPanel.add(tabbedPane);

mainPanel.add(Box.createVerticalStrut(20));

// 3. Panel de botones

JPanel buttonPanel = new JPanel();

buttonPanel.setBackground(new Color(240, 245, 250));

buttonPanel.setBorder(BorderFactory.createEmptyBorder(10, 0, 0, 0));

// Botón Editar

btnEditar = new JButton("Editar");

styleButton(btnEditar, new Color(70, 130, 180));

btnEditar.addActionListener(e -> editarProducto());

// Botón Eliminar

btnEliminar = new JButton("Eliminar");

styleButton(btnEliminar, new Color(180, 70, 70));

btnEliminar.addActionListener(e -> eliminarProducto());

// Botón Contactar Proveedor (solo si hay proveedor)

if (producto.getProveedor() != null && !producto.getProveedor().isEmpty()) {

btnContactarProveedor = new JButton("Contactar Proveedor");

styleButton(btnContactarProveedor, new Color(90, 150, 90));

btnContactarProveedor.addActionListener(e -> {

Proveedor proveedor = proveedorDAO.buscarProveedorPorId(producto.getProveedor());

mostrarInfoProveedor(proveedor);

});

buttonPanel.add(btnContactarProveedor);

buttonPanel.add(Box.createHorizontalStrut(15));

}

// Botón Cerrar

JButton btnCerrar = new JButton("Cerrar");

styleButton(btnCerrar, new Color(100, 100, 100));

btnCerrar.addActionListener(e -> detallesDialog.dispose());

buttonPanel.add(btnEditar);

buttonPanel.add(Box.createHorizontalStrut(15));

buttonPanel.add(btnEliminar);

buttonPanel.add(Box.createHorizontalStrut(15));

buttonPanel.add(btnCerrar);

mainPanel.add(buttonPanel);

detallesDialog.add(mainPanel, BorderLayout.CENTER);

detallesDialog.setVisible(true);

}

// Métodos auxiliares para crear los paneles de cada pestaña

private JPanel createInfoPanel(Producto producto) {

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

panel.setBackground(Color.WHITE);

addDetailRow(panel, "Nombre:", producto.getNombre());

addDetailRow(panel, "ID:", producto.getId());

addDetailRow(panel, "Descripción:", producto.getDescripcion());

addDetailRow(panel, "Categoría:", producto.getCategoria());

addDetailRow(panel, "Unidad de Medida:", producto.getUnidadMedida());

addDetailRow(panel, "Estado:", producto.getEstado());

addDetailRow(panel, "Fecha de Ingreso:",

new SimpleDateFormat("dd/MM/yyyy").format(producto.getFechaIngreso()));

return panel;

}

private JPanel createStockPanel(Producto producto) {

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

panel.setBackground(Color.WHITE);

addDetailRow(panel, "Precio de Compra:", String.format("$%.2f", producto.getPrecioCompra()));

addDetailRow(panel, "Precio de Venta:", String.format("$%.2f", producto.getPrecioVenta()));

if (producto.getDescuento() > 0) {

addDetailRow(panel, "Descuento:", String.format("%.0f%%", producto.getDescuento()));

addDetailRow(panel, "Precio con Descuento:",

String.format("$%.2f", producto.getPrecioConDescuento()));

}

addDetailRow(panel, "IVA:", producto.isTieneIVA() ? "Sí (16%)" : "No");

if (producto.isTieneIVA()) {

addDetailRow(panel, "Precio con IVA:",

String.format("$%.2f", producto.getPrecioConIVA()));

}

addDetailRow(panel, "Stock Actual:",

formatStockValue(producto.getCantidadDisponible(), producto.getUnidadMedida(),

producto.necesitaReposicion()));

addDetailRow(panel, "Stock Mínimo:",

String.valueOf(producto.getStockMinimo()) + " " + producto.getUnidadMedida());

addDetailRow(panel, "Stock Máximo:",

String.valueOf(producto.getStockMaximo()) + " " + producto.getUnidadMedida());

// Añadir alerta si es necesario

if (producto.necesitaReposicion()) {

JLabel alertLabel = new JLabel("¡NECESITA REPOSICIÓN!");

alertLabel.setFont(new Font("Segoe UI", Font.BOLD, 12));

alertLabel.setForeground(Color.RED);

alertLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(Box.createVerticalStrut(15));

panel.add(alertLabel);

} else if (producto.tieneExcesoStock()) {

JLabel alertLabel = new JLabel("¡EXCESO DE STOCK!");

alertLabel.setFont(new Font("Segoe UI", Font.BOLD, 12));

alertLabel.setForeground(new Color(255, 140, 0));

alertLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(Box.createVerticalStrut(15));

panel.add(alertLabel);

}

return panel;

}

private JPanel createSupplierPanel(Producto producto) {

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

panel.setBackground(Color.WHITE);

Proveedor proveedor = proveedorDAO.buscarProveedorPorId(producto.getProveedor());

if (proveedor != null) {

addDetailRow(panel, "Proveedor:", proveedor.getNombre());

addDetailRow(panel, "ID Proveedor:", proveedor.getId());

addDetailRow(panel, "Teléfono:", proveedor.getTelefono());

addDetailRow(panel, "Dirección:",

proveedor.getDireccion() != null ? proveedor.getDireccion() : "No disponible");

addDetailRow(panel, "Productos Suministrados:",

proveedor.getProductoSuministrado() != null ? proveedor.getProductoSuministrado() : "No disponible");

addDetailRow(panel, "Última Visita:",

proveedor.getUltimaVisita() != null ?

new SimpleDateFormat("dd/MM/yyyy").format(proveedor.getUltimaVisita()) : "No registrada");

} else {

JLabel noInfoLabel = new JLabel("No se encontró información del proveedor");

noInfoLabel.setFont(new Font("Segoe UI", Font.ITALIC, 12));

noInfoLabel.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(noInfoLabel);

}

return panel;

}

// Métodos auxiliares

private void addDetailRow(JPanel panel, String label, String value) {

JPanel row = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 5));

row.setBackground(Color.WHITE);

JLabel lbl = new JLabel(label);

lbl.setFont(new Font("Segoe UI", Font.BOLD, 12));

lbl.setPreferredSize(new Dimension(150, 20));

JLabel val = new JLabel(value != null ? value : "N/A");

val.setFont(new Font("Segoe UI", Font.PLAIN, 12));

row.add(lbl);

row.add(val);

panel.add(row);

panel.add(Box.createVerticalStrut(5));

}

private String formatStockValue(int cantidad, String unidad, boolean bajoStock) {

String texto = cantidad + " " + unidad;

if (bajoStock) {

return "<html><font color='red'>" + texto + " (BAJO STOCK)</font></html>";

}

return texto;

}

private void styleButton(JButton button, Color color) {

button.setBackground(color);

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createEmptyBorder(8, 20, 8, 20));

button.setFont(new Font("Segoe UI", Font.BOLD, 12));

button.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

button.setBackground(color.darker());

}

@Override

public void mouseExited(MouseEvent e) {

button.setBackground(color);

}

});

}

private BufferedImage createPlaceholderImage(String text, int width, int height) {

BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

Graphics2D g2d = image.createGraphics();

// Fondo

g2d.setColor(new Color(240, 240, 240));

g2d.fillRect(0, 0, width, height);

// Rectángulo central

g2d.setColor(new Color(220, 230, 240));

g2d.fillRoundRect(10, 10, width-20, height-20, 20, 20);

// Texto

g2d.setColor(new Color(80, 80, 80));

g2d.setFont(new Font("Segoe UI", Font.PLAIN, 14));

// Centrar texto

FontMetrics fm = g2d.getFontMetrics();

String[] lines = text.split("\n");

int lineHeight = fm.getHeight();

int y = (height - (lines.length \* lineHeight)) / 2 + fm.getAscent();

for (String line : lines) {

int textWidth = fm.stringWidth(line);

int x = (width - textWidth) / 2;

g2d.drawString(line, x, y);

y += lineHeight;

}

g2d.dispose();

return image;

}

private void mostrarInfoProveedor(Proveedor proveedor) {

JDialog proveedorDialog = new JDialog(this, "Información del Proveedor", true);

proveedorDialog.setSize(500, 300);

proveedorDialog.setLocationRelativeTo(this);

JPanel panel = new JPanel(new GridLayout(0, 1, 10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

if (proveedor == null) {

panel.add(new JLabel("No hay información disponible del proveedor", JLabel.CENTER));

} else {

panel.add(new JLabel("Nombre: " + proveedor.getNombre()));

panel.add(new JLabel("Teléfono: " + (proveedor.getTelefono() != null ? proveedor.getTelefono() : "No disponible")));

panel.add(new JLabel("Dirección: " + (proveedor.getDireccion() != null ? proveedor.getDireccion() : "No disponible")));

panel.add(new JLabel("Productos suministrados: " + (proveedor.getProductoSuministrado() != null ? proveedor.getProductoSuministrado() : "No disponible")));

panel.add(new JLabel("Última visita: " + (proveedor.getUltimaVisita() != null ? proveedor.getUltimaVisita().toString() : "No disponible")));

}

JButton btnCerrar = new JButton("Cerrar");

btnCerrar.addActionListener(e -> proveedorDialog.dispose());

panel.add(btnCerrar);

proveedorDialog.getContentPane().add(panel);

proveedorDialog.setVisible(true);

}

private void editarProducto() {

if (productoSeleccionado == null) {

JOptionPane.showMessageDialog(this, "Seleccione un producto primero", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

try {

agregarinventario editarDialog = new agregarinventario(usuario, this);

editarDialog.setTitle("Editar Producto: " + productoSeleccionado.getNombre());

editarDialog.setSize(900, 750);

editarDialog.setLocationRelativeTo(this);

editarDialog.cargarDatosProducto(productoSeleccionado);

editarDialog.setGuardarListener(e -> {

try {

Producto productoEditado = editarDialog.obtenerProductoDelFormulario();

productoEditado.setId(productoSeleccionado.getId());

productoEditado.setFechaIngreso(productoSeleccionado.getFechaIngreso());

// Validación adicional

if (productoEditado.getPrecioVenta() <= productoEditado.getPrecioCompra()) {

throw new IllegalArgumentException("El precio de venta debe ser mayor al de compra");

}

if (controlador.actualizarProducto(productoEditado)) {

JOptionPane.showMessageDialog(editarDialog, "Producto actualizado", "Éxito", JOptionPane.INFORMATION\_MESSAGE);

editarDialog.dispose();

if (detallesDialog != null) detallesDialog.dispose();

mostrarProductos(controlador.getTodosProductos());

} else {

throw new Exception("No se pudo actualizar el producto en la base de datos");

}

} catch (Exception ex) {

JOptionPane.showMessageDialog(editarDialog,

"Error al guardar: " + ex.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

ex.printStackTrace();

}

});

editarDialog.setVisible(true);

} catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Error al iniciar edición: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

e.printStackTrace();

}

}

private void eliminarProducto() {

if (productoSeleccionado == null) {

JOptionPane.showMessageDialog(this, "Seleccione un producto primero", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int confirm = JOptionPane.showConfirmDialog(

this,

"¿Está seguro que desea eliminar el producto " + productoSeleccionado.getNombre() + "?",

"Confirmar Eliminación",

JOptionPane.YES\_NO\_OPTION

);

if (confirm == JOptionPane.YES\_OPTION) {

if (controlador.eliminarProducto(productoSeleccionado.getId())) {

JOptionPane.showMessageDialog(this, "Producto eliminado correctamente", "Éxito", JOptionPane.INFORMATION\_MESSAGE);

detallesDialog.dispose();

mostrarProductos(controlador.getTodosProductos());

} else {

JOptionPane.showMessageDialog(this, "Error al eliminar el producto", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

private Image createSampleProductImage() {

int width = 150, height = 150;

BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

Graphics2D g2d = image.createGraphics();

// Fondo blanco

g2d.setColor(Color.WHITE);

g2d.fillRect(0, 0, width, height);

// Dibuja un rectángulo con el nombre como placeholder

g2d.setColor(new Color(200, 200, 255));

g2d.fillRoundRect(10, 10, width - 20, height - 20, 20, 20);

g2d.setColor(Color.BLUE);

g2d.setFont(new Font("Arial", Font.BOLD, 14));

g2d.drawString("Imagen", 50, 80);

g2d.dispose();

return image;

}

private void cambiarUsuario() {

JDialog changeUserDialog = new JDialog(this, "Cambiar Usuario", true);

changeUserDialog.setSize(300, 150);

changeUserDialog.setLocationRelativeTo(this);

JPanel dialogPanel = new JPanel(new BorderLayout(10, 10));

dialogPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

JLabel instructionLabel = new JLabel("¿Desea cambiar de usuario?", SwingConstants.CENTER);

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 0));

JButton cambiarBtn = new JButton("Cambiar de Usuario");

JButton cancelarBtn = new JButton("Cancelar");

cambiarBtn.addActionListener(e -> {

changeUserDialog.dispose();

this.dispose();

new Login().setVisible(true);

});

cancelarBtn.addActionListener(e -> changeUserDialog.dispose());

buttonPanel.add(cambiarBtn);

buttonPanel.add(cancelarBtn);

dialogPanel.add(instructionLabel, BorderLayout.CENTER);

dialogPanel.add(buttonPanel, BorderLayout.SOUTH);

changeUserDialog.getContentPane().add(dialogPanel);

changeUserDialog.setVisible(true);

}

private JButton crearBotonMenu(String texto) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

boton.setPreferredSize(new Dimension(0, 50));

boton.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 50));

boton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseEntered(java.awt.event.MouseEvent evt) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

public void mouseExited(java.awt.event.MouseEvent evt) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

});

return boton;

}

private void manejarAccionMenu(String opcion) {

switch (opcion) {

case "Salir":

this.dispose();

new menuprincipal(usuario).setVisible(true);

break;

case "Productos":

this.dispose();

new producto(usuario).setVisible(true);

break;

case "Reportes":

this.dispose();

reportes vistaReportes = new reportes(usuario, new ReportesControlador(null, usuario));

vistaReportes.setVisible(true); // Muestra la ventana

break;

case "Inventario":

JOptionPane.showMessageDialog(this, "Ya estás en la ventana de Inventario.");

break;

case "Cliente":

this.dispose();

ClienteDAOImpl clienteDAO = new ClienteDAOImpl();

new clientes(usuario, clienteDAO).setVisible(true);

break;

case "Proveedores":

this.dispose();

new proveedores(usuario).setVisible(true);

break;

case "Usuario":

this.dispose();

new gestionUsuario(usuario).setVisible(true);

break;

}

}

private void agregarProducto() {

agregarinventario agregar = new agregarinventario(usuario, this);

agregar.setVisible(true);

// Actualizar la vista después de cerrar el diálogo

mostrarProductos(controlador.getTodosProductos());

}

private void generarReporte() {

// Implementar generación de reportes

JOptionPane.showMessageDialog(this, "Generando reporte...", "Reporte", JOptionPane.INFORMATION\_MESSAGE);

}

public void setControlador(ControladorInventario controlador) {

this.controlador = controlador;

}

private JButton createCategoryButton(String text) {

JButton button = new JButton(text);

button.setAlignmentX(Component.CENTER\_ALIGNMENT);

button.setMaximumSize(new Dimension(180, 40));

button.setPreferredSize(new Dimension(180, 40));

// Estilo del botón

button.setBackground(new Color(255, 182, 193)); // Rosa claro

button.setForeground(Color.BLACK);

button.setFont(new Font("Arial", Font.BOLD, 12));

button.setFocusPainted(false);

button.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(new Color(200, 120, 150)), // Borde rosado oscuro

BorderFactory.createEmptyBorder(5, 10, 5, 10)

));

// Efecto hover

button.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

button.setBackground(new Color(255, 160, 180)); // Rosa más intenso

}

@Override

public void mouseExited(MouseEvent e) {

button.setBackground(new Color(255, 182, 193)); // Volver al color original

}

});

return button;

}

// Método para botones con estilo moderno

private JButton createStyledButton(String text, Color bgColor) {

JButton button = new JButton(text);

button.setBackground(bgColor);

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createEmptyBorder(8, 15, 8, 15));

button.setFont(new Font("Arial", Font.BOLD, 12));

button.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

button.setBackground(bgColor.darker());

}

@Override

public void mouseExited(MouseEvent e) {

button.setBackground(bgColor);

}

});

return button;

}

// Método modificado para el panel de categorías

private JPanel createCategoryFilterPanel() {

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

panel.setBackground(new Color(245, 245, 245));

// Título

JLabel lblTitle = new JLabel("CATEGORÍAS:");

lblTitle.setFont(new Font("Segoe UI", Font.BOLD, 12));

lblTitle.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(lblTitle);

panel.add(Box.createVerticalStrut(10));

// Panel de categorías con scroll (solo si es necesario)

JPanel categoriesPanel = new JPanel();

categoriesPanel.setLayout(new BoxLayout(categoriesPanel, BoxLayout.Y\_AXIS));

categoriesPanel.setBackground(new Color(245, 245, 245));

JScrollPane scrollPane = new JScrollPane(categoriesPanel);

scrollPane.setBorder(null);

scrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL\_SCROLLBAR\_AS\_NEEDED);

scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_NEVER);

panel.add(scrollPane);

// Botón "Todas"

JButton btnAll = createRoundButton("TODAS LAS CATEGORÍAS", Color.PINK);

btnAll.setAlignmentX(Component.CENTER\_ALIGNMENT);

btnAll.addActionListener(e -> mostrarProductos(controlador.getTodosProductos()));

categoriesPanel.add(btnAll);

categoriesPanel.add(Box.createVerticalStrut(10));

// Cargar categorías desde BD

loadCategoriesFromDB(categoriesPanel);

// Panel para agregar (solo admin)

if (usuario.esAdministrador()) {

panel.add(Box.createVerticalStrut(15));

JPanel addPanel = createAddCategoryPanel(categoriesPanel);

panel.add(addPanel);

}

return panel;

}

// Método para cargar categorías desde BD

private void loadCategoriesFromDB(JPanel container) {

container.removeAll(); // Limpiar primero

List<String> categorias = controlador.getCategorias();

System.out.println("Categorías cargadas desde BD: " + categorias); // Debug

if (categorias.isEmpty()) {

JLabel lblEmpty = new JLabel("No hay categorías registradas");

lblEmpty.setAlignmentX(Component.CENTER\_ALIGNMENT);

container.add(lblEmpty);

} else {

for (String categoria : categorias) {

JButton btnCat = createRoundButton(categoria, new Color(120, 120, 120));

btnCat.setAlignmentX(Component.CENTER\_ALIGNMENT);

btnCat.addActionListener(e -> mostrarProductos(controlador.buscarPorCategoria(categoria)));

container.add(btnCat);

container.add(Box.createVerticalStrut(8));

}

}

container.revalidate();

container.repaint();

}

// Método para crear botones redondeados (como en tu diseño anterior)

private JButton createRoundButton(String text, Color bgColor) {

JButton button = new JButton(text) {

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(getBackground());

g2.fillRoundRect(0, 0, getWidth(), getHeight(), 25, 25);

super.paintComponent(g2);

g2.dispose();

}

@Override

protected void paintBorder(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(getBackground().darker());

g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 25, 25);

g2.dispose();

}

};

button.setOpaque(false);

button.setContentAreaFilled(false);

button.setBorderPainted(false);

button.setBackground(Color.PINK);

button.setForeground(Color.BLACK);

button.setFont(new Font("Segoe UI", Font.BOLD, 14));

button.setFocusPainted(false);

button.setPreferredSize(new Dimension(180, 40));

button.setMaximumSize(new Dimension(180, 40));

button.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

button.setBackground(button.getBackground().brighter());

}

@Override

public void mouseExited(MouseEvent e) {

button.setBackground(Color.PINK);

}

});

return button;

}

// Panel para agregar nuevas categorías (conservando tu diseño)

private JPanel createAddCategoryPanel(JPanel categoriesPanel) {

JPanel panel = new JPanel();

panel.setLayout(new BoxLayout(panel, BoxLayout.X\_AXIS));

panel.setBackground(new Color(245, 245, 245));

panel.setBorder(BorderFactory.createEmptyBorder(5, 20, 5, 20));

JTextField txtNewCat = new JTextField();

txtNewCat.setMaximumSize(new Dimension(150, 30));

txtNewCat.setFont(new Font("Segoe UI", Font.PLAIN, 12));

JButton btnAdd = new JButton("+");

btnAdd.setFont(new Font("Segoe UI", Font.BOLD, 14));

btnAdd.setBackground(new Color(80, 180, 80));

btnAdd.setForeground(Color.WHITE);

btnAdd.setBorder(BorderFactory.createEmptyBorder(5, 15, 5, 15));

btnAdd.addActionListener(e -> {

String newCategory = txtNewCat.getText().trim();

if (!newCategory.isEmpty()) {

if (controlador.agregarCategoria(newCategory, "")) {

txtNewCat.setText("");

loadCategoriesFromDB(categoriesPanel); // Recargar categorías

} else {

JOptionPane.showMessageDialog(panel, "Error al agregar categoría");

}

}

});

panel.add(txtNewCat);

panel.add(Box.createHorizontalStrut(10));

panel.add(btnAdd);

return panel;

}

// Método optimizado para cargar categorías

private void loadCategories(JPanel container) {

container.removeAll();

// Botón "Todas"

JButton btnAll = createCategoryButton("TODAS", Color.PINK);

btnAll.addActionListener(e -> mostrarProductos(controlador.getTodosProductos()));

container.add(btnAll);

// Categorías existentes

for (String categoria : controlador.getCategorias()) {

JButton btnCat = createCategoryButton(categoria, Color.PINK);

btnCat.addActionListener(e -> mostrarProductos(controlador.buscarPorCategoria(categoria)));

container.add(btnCat);

}

container.revalidate();

container.repaint();

}

// Botón de categoría optimizado

private JButton createCategoryButton(String text, Color bgColor) {

JButton button = new JButton(text.length() > 12 ? text.substring(0, 10) + "..." : text);

button.setFont(new Font("Arial", Font.PLAIN, 11));

button.setBackground(bgColor);

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createEmptyBorder(5, 8, 5, 8));

button.setToolTipText(text); // Muestra texto completo al pasar el mouse

button.setPreferredSize(new Dimension(100, 30));

button.addMouseListener(new MouseAdapter() {

public void mouseEntered(MouseEvent e) {

button.setBackground(bgColor.brighter());

}

public void mouseExited(MouseEvent e) {

button.setBackground(bgColor);

}

});

return button;

}

// Método para agregar nueva categoría

private void addNewCategory(JTextField textField, JPanel categoriesPanel) {

String newCat = textField.getText().trim();

if (!newCat.isEmpty()) {

if (controlador.agregarCategoria(newCat, "")) {

textField.setText("");

loadCategories(categoriesPanel);

} else {

JOptionPane.showMessageDialog(panel, "Error al agregar categoría");

}

}

}

// Clase WrapLayout para flujo automático (añadir al proyecto)

public class WrapLayout extends FlowLayout {

public WrapLayout() { super(); }

public WrapLayout(int align) { super(align); }

public WrapLayout(int align, int hgap, int vgap) { super(align, hgap, vgap); }

@Override

public Dimension preferredLayoutSize(Container target) {

return layoutSize(target, true);

}

@Override

public Dimension minimumLayoutSize(Container target) {

return layoutSize(target, false);

}

private Dimension layoutSize(Container target, boolean preferred) {

synchronized (target.getTreeLock()) {

int targetWidth = target.getSize().width;

if (targetWidth == 0) targetWidth = Integer.MAX\_VALUE;

int hgap = getHgap();

int vgap = getVgap();

Insets insets = target.getInsets();

int maxWidth = targetWidth - (insets.left + insets.right + hgap \* 2);

Dimension dim = new Dimension(0, 0);

int rowWidth = 0, rowHeight = 0;

for (Component m : target.getComponents()) {

if (m.isVisible()) {

Dimension d = preferred ? m.getPreferredSize() : m.getMinimumSize();

if (rowWidth + d.width > maxWidth) {

dim.width = Math.max(dim.width, rowWidth);

dim.height += rowHeight + vgap;

rowWidth = 0;

rowHeight = 0;

}

rowWidth += d.width + hgap;

rowHeight = Math.max(rowHeight, d.height);

}

}

dim.width = Math.max(dim.width, rowWidth);

dim.height += rowHeight;

dim.width += insets.left + insets.right + hgap \* 2;

dim.height += insets.top + insets.bottom + vgap \* 2;

return dim;

}

}

}

private void cargarCategorias(JPanel container) {

container.removeAll();

container.setLayout(new GridBagLayout()); // Cambia a GridBagLayout

GridBagConstraints gbc = new GridBagConstraints();

gbc.gridwidth = GridBagConstraints.REMAINDER;

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.insets = new Insets(5, 5, 5, 5);

List<String> categorias = controlador.getCategorias();

for (String categoria : categorias) {

JPanel categoriaPanel = new JPanel(new BorderLayout());

categoriaPanel.setBackground(new Color(240, 240, 240));

// Botón para filtrar

JButton btnCategoria = createStyledButton(categoria, new Color(120, 120, 120)); btnCategoria.addActionListener(e ->

mostrarProductos(controlador.buscarPorCategoria(categoria)));

// Botón eliminar (solo admin)

if (usuario.esAdministrador()) {

JButton btnEliminar = createStyledButton("X", new Color(200, 80, 80));

btnEliminar.setPreferredSize(new Dimension(30, 20));

btnEliminar.addActionListener(e -> {

if (JOptionPane.showConfirmDialog(container,

"¿Eliminar categoría " + categoria + "?",

"Confirmar", JOptionPane.YES\_NO\_OPTION) == JOptionPane.YES\_OPTION) {

if (controlador.eliminarCategoria(categoria)) {

cargarCategorias(container); // Recargar

}

}

});

container.add(btnCategoria, gbc);

categoriaPanel.add(btnEliminar, BorderLayout.EAST);

} else {

container.add(btnCategoria, gbc);

}

container.add(categoriaPanel);

container.add(Box.createVerticalStrut(5));

}

container.revalidate();

container.repaint();

}

// Método para la barra inferior

private JPanel createBottomPanel() {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 10));

panel.setBackground(new Color(240, 240, 240));

panel.setBorder(BorderFactory.createEmptyBorder(10, 20, 15, 20));

// Botón Agregar

JButton btnAgregar = createRoundButton("AGREGAR", new Color(120, 120, 120), 25);

btnAgregar.addActionListener(e -> agregarProducto());

// Botón Búsqueda Avanzada

JButton btnAvanzada = createRoundButton("BÚSQUEDA AVANZADA", new Color(120, 120, 120), 25);

btnAvanzada.addActionListener(e -> abrirBusquedaAvanzada());

// Botón Reporte

JButton btnReporte = createRoundButton("REPORTE", new Color(120, 120, 120), 25);

btnReporte.addActionListener(e -> abrirReporteInventario());

panel.add(btnAgregar);

panel.add(btnAvanzada);

panel.add(btnReporte);

return panel;

}

private JButton createRoundButton(String text, Color bgColor, int arc) {

JButton button = new JButton(text) {

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Fondo redondeado

g2.setColor(bgColor);

g2.fillRoundRect(0, 0, getWidth(), getHeight(), arc, arc);

// Texto

g2.setColor(Color.WHITE);

FontMetrics fm = g2.getFontMetrics();

Rectangle2D r = fm.getStringBounds(text, g2);

int x = (getWidth() - (int) r.getWidth()) / 2;

int y = (getHeight() - (int) r.getHeight()) / 2 + fm.getAscent();

g2.drawString(text, x, y);

g2.dispose();

}

@Override

protected void paintBorder(Graphics g) {

// Sin borde

}

};

button.setContentAreaFilled(false);

button.setFocusPainted(false);

button.setBorderPainted(false);

button.setFont(new Font("Segoe UI", Font.BOLD, 12));

button.setPreferredSize(new Dimension(150, 40));

button.setCursor(new Cursor(Cursor.HAND\_CURSOR));

// Efecto hover

button.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

button.setBackground(bgColor.brighter());

}

@Override

public void mouseExited(MouseEvent e) {

button.setBackground(bgColor);

}

});

return button;

}

private JButton createModernButton(String text, Color bgColor, Color textColor) {

JButton button = new JButton(text);

button.setFont(new Font("Segoe UI", Font.BOLD, 12));

button.setBackground(bgColor);

button.setForeground(textColor);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(bgColor.darker(), 1),

BorderFactory.createEmptyBorder(8, 20, 8, 20)

));

button.setCursor(new Cursor(Cursor.HAND\_CURSOR));

// Efecto hover

button.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

button.setBackground(bgColor.brighter());

button.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(bgColor.darker(), 1),

BorderFactory.createEmptyBorder(8, 20, 8, 20)

));

}

@Override

public void mouseExited(MouseEvent e) {

button.setBackground(bgColor);

button.setBorder(BorderFactory.createCompoundBorder(

BorderFactory.createLineBorder(bgColor.darker(), 1),

BorderFactory.createEmptyBorder(8, 20, 8, 20)

));

}

});

return button;

}

private void abrirBusquedaAvanzada() {

JDialog dialog = new JDialog(this, "Búsqueda Avanzada de Productos", true);

dialog.setSize(900, 700);

dialog.setLocationRelativeTo(this);

dialog.setLayout(new BorderLayout());

dialog.getContentPane().setBackground(new Color(240, 245, 250));

// Panel principal con pestañas

JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP, JTabbedPane.WRAP\_TAB\_LAYOUT);

tabbedPane.setFont(new Font("Segoe UI", Font.BOLD, 12));

// 1. Pestaña de Búsqueda

tabbedPane.addTab("🔍 Búsqueda", crearPanelBusquedaAvanzada());

// 2. Pestaña de Movimientos

tabbedPane.addTab("📦 Movimientos", crearPanelMovimientos());

// 3. Pestaña de Historial

tabbedPane.addTab("📜 Historial", crearPanelHistorial());

// 4. pestaña de Devoluciones

tabbedPane.addTab("🔄 Devoluciones", crearPanelDevoluciones());

// Panel inferior con botones

JPanel bottomPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15, 10));

bottomPanel.setBorder(BorderFactory.createEmptyBorder(10, 0, 15, 0));

bottomPanel.setBackground(new Color(240, 245, 250));

JButton btnBuscar = createModernButton("Buscar", new Color(70, 130, 180), Color.WHITE);

btnBuscar.setPreferredSize(new Dimension(120, 35));

btnBuscar.addActionListener(e -> realizarBusquedaAvanzada());

JButton btnLimpiar = createModernButton("Limpiar", new Color(120, 120, 120), Color.WHITE);

btnLimpiar.setPreferredSize(new Dimension(120, 35));

btnLimpiar.addActionListener(e -> limpiarCamposBusqueda());

JButton btnCerrar = createModernButton("Cerrar", new Color(192, 57, 43), Color.WHITE);

btnCerrar.setPreferredSize(new Dimension(120, 35));

btnCerrar.addActionListener(e -> dialog.dispose());

bottomPanel.add(btnBuscar);

bottomPanel.add(btnLimpiar);

bottomPanel.add(btnCerrar);

dialog.add(tabbedPane, BorderLayout.CENTER);

dialog.add(bottomPanel, BorderLayout.SOUTH);

dialog.setVisible(true);

}

private JPanel crearPanelBusquedaAvanzada() {

JPanel panel = new JPanel(new GridBagLayout());

panel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

panel.setBackground(Color.WHITE);

GridBagConstraints gbc = new GridBagConstraints();

gbc.anchor = GridBagConstraints.WEST;

gbc.insets = new Insets(8, 8, 8, 8);

gbc.fill = GridBagConstraints.HORIZONTAL;

// 1. Fila - Nombre

gbc.gridx = 0;

gbc.gridy = 0;

panel.add(createStyledLabel("Nombre:"), gbc);

gbc.gridx = 1;

JTextField txtNombre = new JTextField(20);

txtNombre.setFont(new Font("Segoe UI", Font.PLAIN, 14));

panel.add(txtNombre, gbc);

// 2. Fila - Categoría

gbc.gridx = 0;

gbc.gridy = 1;

panel.add(createStyledLabel("Categoría:"), gbc);

gbc.gridx = 1;

JComboBox<String> cbCategorias = new JComboBox<>();

cbCategorias.setFont(new Font("Segoe UI", Font.PLAIN, 14));

cbCategorias.addItem("Todas las categorías");

controlador.getCategorias().forEach(cbCategorias::addItem);

panel.add(cbCategorias, gbc);

// 3. Fila - Proveedor

gbc.gridx = 0;

gbc.gridy = 2;

panel.add(createStyledLabel("Proveedor:"), gbc);

gbc.gridx = 1;

JComboBox<String> cbProveedores = new JComboBox<>();

cbProveedores.setFont(new Font("Segoe UI", Font.PLAIN, 14));

cbProveedores.addItem("Todos los proveedores");

proveedorDAO.obtenerTodosProveedores().forEach(p -> cbProveedores.addItem(p.getNombre()));

panel.add(cbProveedores, gbc);

// 4. Fila - Estado

gbc.gridx = 0;

gbc.gridy = 3;

panel.add(createStyledLabel("Estado:"), gbc);

gbc.gridx = 1;

JComboBox<String> cbEstado = new JComboBox<>(new String[]{"Todos", "Activo", "Descontinuado", "Dañado"});

cbEstado.setFont(new Font("Segoe UI", Font.PLAIN, 14));

panel.add(cbEstado, gbc);

// 5. Fila - Rango de Precios

gbc.gridx = 0;

gbc.gridy = 4;

panel.add(createStyledLabel("Rango de Precio:"), gbc);

gbc.gridx = 1;

JPanel panelPrecio = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));

panelPrecio.setBackground(Color.WHITE);

JFormattedTextField txtPrecioMin = new JFormattedTextField(NumberFormat.getCurrencyInstance());

txtPrecioMin.setColumns(8);

txtPrecioMin.setFont(new Font("Segoe UI", Font.PLAIN, 14));

JLabel lblHasta = new JLabel(" hasta ");

lblHasta.setFont(new Font("Segoe UI", Font.PLAIN, 14));

JFormattedTextField txtPrecioMax = new JFormattedTextField(NumberFormat.getCurrencyInstance());

txtPrecioMax.setColumns(8);

txtPrecioMax.setFont(new Font("Segoe UI", Font.PLAIN, 14));

panelPrecio.add(txtPrecioMin);

panelPrecio.add(lblHasta);

panelPrecio.add(txtPrecioMax);

panel.add(panelPrecio, gbc);

// 6. Fila - Rango de Stock

gbc.gridx = 0;

gbc.gridy = 5;

panel.add(createStyledLabel("Rango de Stock:"), gbc);

gbc.gridx = 1;

JPanel panelStock = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));

panelStock.setBackground(Color.WHITE);

JSpinner spnStockMin = new JSpinner(new SpinnerNumberModel(0, 0, 10000, 1));

spnStockMin.setFont(new Font("Segoe UI", Font.PLAIN, 14));

JLabel lblHastaStock = new JLabel(" hasta ");

lblHastaStock.setFont(new Font("Segoe UI", Font.PLAIN, 14));

JSpinner spnStockMax = new JSpinner(new SpinnerNumberModel(100, 0, 10000, 1));

spnStockMax.setFont(new Font("Segoe UI", Font.PLAIN, 14));

panelStock.add(spnStockMin);

panelStock.add(lblHastaStock);

panelStock.add(spnStockMax);

panel.add(panelStock, gbc);

// 7. Fila - Fechas

gbc.gridx = 0;

gbc.gridy = 6;

panel.add(createStyledLabel("Fecha de Ingreso:"), gbc);

gbc.gridx = 1;

JPanel panelFechas = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));

panelFechas.setBackground(Color.WHITE);

// Crear JSpinner para fecha desde

JSpinner spnFechaDesde = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor fechaEditorDesde = new JSpinner.DateEditor(spnFechaDesde, "dd/MM/yyyy");

spnFechaDesde.setEditor(fechaEditorDesde);

spnFechaDesde.setFont(new Font("Segoe UI", Font.PLAIN, 14));

JLabel lblHastaFecha = new JLabel(" hasta ");

lblHastaFecha.setFont(new Font("Segoe UI", Font.PLAIN, 14));

// Crear JSpinner para fecha hasta

JSpinner spnFechaHasta = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor fechaEditorHasta = new JSpinner.DateEditor(spnFechaHasta, "dd/MM/yyyy");

spnFechaHasta.setEditor(fechaEditorHasta);

spnFechaHasta.setFont(new Font("Segoe UI", Font.PLAIN, 14));

panelFechas.add(spnFechaDesde);

panelFechas.add(lblHastaFecha);

panelFechas.add(spnFechaHasta);

panel.add(panelFechas, gbc);

// 8. Fila - Checkboxes adicionales

gbc.gridx = 0;

gbc.gridy = 7;

gbc.gridwidth = 2;

JPanel panelChecks = new JPanel(new GridLayout(1, 3, 10, 0));

panelChecks.setBackground(Color.WHITE);

JCheckBox chkNecesitaReposicion = new JCheckBox("Necesita reposición");

styleCheckbox(chkNecesitaReposicion);

JCheckBox chkTieneDescuento = new JCheckBox("Tiene descuento");

styleCheckbox(chkTieneDescuento);

JCheckBox chkTieneIVA = new JCheckBox("Aplica IVA");

styleCheckbox(chkTieneIVA);

panelChecks.add(chkNecesitaReposicion);

panelChecks.add(chkTieneDescuento);

panelChecks.add(chkTieneIVA);

panel.add(panelChecks, gbc);

return panel;

}

private JLabel createStyledLabel(String text) {

JLabel label = new JLabel(text);

label.setFont(new Font("Segoe UI", Font.BOLD, 14));

label.setForeground(new Color(70, 70, 70));

return label;

}

private void styleCheckbox(JCheckBox checkbox) {

checkbox.setFont(new Font("Segoe UI", Font.PLAIN, 13));

checkbox.setBackground(Color.WHITE);

checkbox.setFocusPainted(false);

}

private void realizarBusquedaAvanzada() {

// Implementar lógica de búsqueda avanzada aquí

// Recopilar todos los criterios de búsqueda y llamar al controlador

JOptionPane.showMessageDialog(this,

"Búsqueda avanzada realizada con los criterios seleccionados",

"Resultados",

JOptionPane.INFORMATION\_MESSAGE);

}

private void limpiarCamposBusqueda() {

// Implementar lógica para limpiar todos los campos de búsqueda

JOptionPane.showMessageDialog(this,

"Todos los campos de búsqueda han sido limpiados",

"Campos limpiados",

JOptionPane.INFORMATION\_MESSAGE);

}

private JPanel crearPanelMovimientos() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

JTabbedPane tabsMovimientos = new JTabbedPane();

tabsMovimientos.addTab("➕ Entradas", crearPanelEntradas());

tabsMovimientos.addTab("➖ Salidas", crearPanelSalidas());

tabsMovimientos.addTab("🔄 Ajustes", crearPanelAjustes());

panel.add(tabsMovimientos, BorderLayout.CENTER);

return panel;

}

private JPanel crearPanelHistorial() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// Panel de búsqueda

JPanel searchPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 10));

searchPanel.add(new JLabel("Teléfono cliente:"));

JTextField txtTelefono = new JTextField(15);

searchPanel.add(txtTelefono);

JButton btnBuscar = createStyledButton("Buscar", new Color(70, 130, 180));

btnBuscar.addActionListener(e -> {

// Implementar búsqueda de historial

});

searchPanel.add(btnBuscar);

// Tabla de resultados

String[] columnas = {"Fecha", "Producto", "Cantidad", "Total", "Puntos"};

Object[][] datos = {}; // Datos reales irían aquí

JTable tabla = new JTable(datos, columnas);

panel.add(searchPanel, BorderLayout.NORTH);

panel.add(new JScrollPane(tabla), BorderLayout.CENTER);

return panel;

}

private JPanel crearPanelEntradas() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Panel de formulario

JPanel formPanel = new JPanel(new GridLayout(4, 2, 10, 10));

// Campo ID Producto

JTextField txtIdProducto = new JTextField();

JButton btnBuscarProducto = createStyledButton("Buscar", new Color(70, 130, 180));

JPanel panelIdProducto = new JPanel(new BorderLayout(5, 5));

panelIdProducto.add(txtIdProducto, BorderLayout.CENTER);

panelIdProducto.add(btnBuscarProducto, BorderLayout.EAST);

formPanel.add(new JLabel("ID Producto:"));

formPanel.add(panelIdProducto);

// Campo Nombre Producto (solo lectura)

JTextField txtNombreProducto = new JTextField();

txtNombreProducto.setEditable(false);

formPanel.add(new JLabel("Nombre Producto:"));

formPanel.add(txtNombreProducto);

// Campo Cantidad

JSpinner spnCantidad = new JSpinner(new SpinnerNumberModel(1, 1, 10000, 1));

formPanel.add(new JLabel("Cantidad:"));

formPanel.add(spnCantidad);

// Campo Fecha

JSpinner spnFecha = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor fechaEditor = new JSpinner.DateEditor(spnFecha, "dd/MM/yyyy");

spnFecha.setEditor(fechaEditor);

spnFecha.setValue(new Date());

formPanel.add(new JLabel("Fecha:"));

formPanel.add(spnFecha);

// Botón Registrar

JButton btnRegistrar = createStyledButton("Registrar Entrada", new Color(63, 142, 77));

btnRegistrar.addActionListener(e -> {

if (validarEntrada(txtIdProducto.getText(), (Integer)spnCantidad.getValue())) {

registrarEntrada(

txtIdProducto.getText(),

txtNombreProducto.getText(),

(Integer)spnCantidad.getValue(),

(Date)spnFecha.getValue()

);

}

});

// Configurar acción del botón buscar

btnBuscarProducto.addActionListener(e -> {

Producto producto = buscarProducto(txtIdProducto.getText());

if (producto != null) {

txtNombreProducto.setText(producto.getNombre());

}

});

// Panel principal

panel.add(formPanel, BorderLayout.CENTER);

panel.add(btnRegistrar, BorderLayout.SOUTH);

return panel;

}

private JPanel crearPanelSalidas() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Panel de formulario

JPanel formPanel = new JPanel(new GridLayout(5, 2, 10, 10));

// Campo ID Producto

JTextField txtIdProducto = new JTextField();

JButton btnBuscarProducto = createStyledButton("Buscar", new Color(70, 130, 180));

JPanel panelIdProducto = new JPanel(new BorderLayout(5, 5));

panelIdProducto.add(txtIdProducto, BorderLayout.CENTER);

panelIdProducto.add(btnBuscarProducto, BorderLayout.EAST);

formPanel.add(new JLabel("ID Producto:"));

formPanel.add(panelIdProducto);

// Campo Nombre Producto

JTextField txtNombreProducto = new JTextField();

txtNombreProducto.setEditable(false);

formPanel.add(new JLabel("Nombre Producto:"));

formPanel.add(txtNombreProducto);

// Campo Cantidad

JSpinner spnCantidad = new JSpinner(new SpinnerNumberModel(1, 1, 10000, 1));

formPanel.add(new JLabel("Cantidad:"));

formPanel.add(spnCantidad);

// Campo Motivo

JComboBox<String> cbMotivo = new JComboBox<>(new String[]{

"Venta directa",

"Venta en línea",

"Muestra comercial",

"Donación",

"Otro"

});

formPanel.add(new JLabel("Motivo:"));

formPanel.add(cbMotivo);

// Campo Observaciones

JTextField txtObservaciones = new JTextField();

formPanel.add(new JLabel("Observaciones:"));

formPanel.add(txtObservaciones);

// Botón Registrar

JButton btnRegistrar = createStyledButton("Registrar Salida", new Color(192, 57, 43));

btnRegistrar.addActionListener(e -> {

if (validarSalida(txtIdProducto.getText(), (Integer)spnCantidad.getValue())) {

registrarSalida(

txtIdProducto.getText(),

txtNombreProducto.getText(),

(Integer)spnCantidad.getValue(),

(String)cbMotivo.getSelectedItem(),

txtObservaciones.getText()

);

}

});

// Configurar acción del botón buscar

btnBuscarProducto.addActionListener(e -> {

Producto producto = buscarProducto(txtIdProducto.getText());

if (producto != null) {

txtNombreProducto.setText(producto.getNombre());

// Mostrar stock actual como máximo en el spinner

spnCantidad.setModel(new SpinnerNumberModel(

1, 1, producto.getCantidadDisponible(), 1));

}

});

// Panel principal

panel.add(formPanel, BorderLayout.CENTER);

panel.add(btnRegistrar, BorderLayout.SOUTH);

return panel;

}

private JPanel crearPanelAjustes() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Panel de formulario

JPanel formPanel = new JPanel(new GridLayout(6, 2, 10, 10));

// Campo ID Producto

JTextField txtIdProducto = new JTextField();

JButton btnBuscarProducto = createStyledButton("Buscar", new Color(70, 130, 180));

JPanel panelIdProducto = new JPanel(new BorderLayout(5, 5));

panelIdProducto.add(txtIdProducto, BorderLayout.CENTER);

panelIdProducto.add(btnBuscarProducto, BorderLayout.EAST);

formPanel.add(new JLabel("ID Producto:"));

formPanel.add(panelIdProducto);

// Campo Nombre Producto

JTextField txtNombreProducto = new JTextField();

txtNombreProducto.setEditable(false);

formPanel.add(new JLabel("Nombre Producto:"));

formPanel.add(txtNombreProducto);

// Campo Tipo de Ajuste

JComboBox<String> cbTipoAjuste = new JComboBox<>(new String[]{

"Merma",

"Daño",

"Robo",

"Error de inventario",

"Caducidad",

"Otro"

});

formPanel.add(new JLabel("Tipo de ajuste:"));

formPanel.add(cbTipoAjuste);

// Campo Cantidad (puede ser positivo o negativo)

JSpinner spnCantidad = new JSpinner(new SpinnerNumberModel(0, -1000, 1000, 1));

formPanel.add(new JLabel("Cantidad:"));

formPanel.add(spnCantidad);

// Campo Fecha

JSpinner spnFecha = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor fechaEditor = new JSpinner.DateEditor(spnFecha, "dd/MM/yyyy");

spnFecha.setEditor(fechaEditor);

spnFecha.setValue(new Date());

formPanel.add(new JLabel("Fecha:"));

formPanel.add(spnFecha);

// Campo Descripción

JTextField txtDescripcion = new JTextField();

formPanel.add(new JLabel("Descripción:"));

formPanel.add(txtDescripcion);

// Botón Registrar

JButton btnRegistrar = createStyledButton("Registrar Ajuste", new Color(243, 156, 18));

btnRegistrar.addActionListener(e -> {

if (validarAjuste(txtIdProducto.getText(), (Integer)spnCantidad.getValue())) {

registrarAjuste(

txtIdProducto.getText(),

txtNombreProducto.getText(),

(String)cbTipoAjuste.getSelectedItem(),

(Integer)spnCantidad.getValue(),

(Date)spnFecha.getValue(),

txtDescripcion.getText()

);

}

});

// Configurar acción del botón buscar

btnBuscarProducto.addActionListener(e -> {

Producto producto = buscarProducto(txtIdProducto.getText());

if (producto != null) {

txtNombreProducto.setText(producto.getNombre());

}

});

// Panel principal

panel.add(formPanel, BorderLayout.CENTER);

panel.add(btnRegistrar, BorderLayout.SOUTH);

return panel;

}

private Producto buscarProducto(String idProducto) {

if (idProducto == null || idProducto.trim().isEmpty()) {

JOptionPane.showMessageDialog(this, "Ingrese un ID de producto", "Error", JOptionPane.ERROR\_MESSAGE);

return null;

}

Producto producto = controlador.getProductoPorId(idProducto);

if (producto == null) {

JOptionPane.showMessageDialog(this, "Producto no encontrado", "Error", JOptionPane.ERROR\_MESSAGE);

}

return producto;

}

private boolean validarEntrada(String idProducto, int cantidad) {

if (idProducto == null || idProducto.trim().isEmpty()) {

JOptionPane.showMessageDialog(this, "Ingrese un ID de producto", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

if (cantidad <= 0) {

JOptionPane.showMessageDialog(this, "La cantidad debe ser mayor a cero", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

private void registrarEntrada(String idProducto, String nombreProducto, int cantidad, Date fecha) {

// Implementar lógica para registrar entrada en la base de datos

JOptionPane.showMessageDialog(this,

String.format("Entrada registrada:\nProducto: %s\nCantidad: %d", nombreProducto, cantidad),

"Éxito", JOptionPane.INFORMATION\_MESSAGE);

}

// Similarmente implementar los métodos:

// validarSalida(), registrarSalida(), validarAjuste(), registrarAjuste()

private boolean validarSalida(String idProducto, int cantidad) {

if (idProducto == null || idProducto.trim().isEmpty()) {

JOptionPane.showMessageDialog(this, "Ingrese un ID de producto", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

if (cantidad <= 0) {

JOptionPane.showMessageDialog(this, "La cantidad debe ser mayor a cero", "Error",

JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

private void registrarSalida(String idProducto, String nombreProducto, int cantidad, String motivo,

String observaciones) {

// Implementar lógica para registrar salida en la base de datos

JOptionPane.showMessageDialog(this, String.format("Salida registrada:\nProducto: %s\nCantidad: %d\nMotivo: %s",

nombreProducto, cantidad, motivo), "Éxito", JOptionPane.INFORMATION\_MESSAGE);

}

private boolean validarAjuste(String idProducto, int cantidad) {

if (idProducto == null || idProducto.trim().isEmpty()) {

JOptionPane.showMessageDialog(this, "Ingrese un ID de producto", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

if (cantidad == 0) {

JOptionPane.showMessageDialog(this, "La cantidad no puede ser cero", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

private void registrarAjuste(String idProducto, String nombreProducto, String tipoAjuste, int cantidad, Date fecha,

String descripcion) {

// Implementar lógica para registrar ajuste en la base de datos

JOptionPane.showMessageDialog(this, String.format("Ajuste registrado:\nProducto: %s\nTipo: %s\nCantidad: %d",

nombreProducto, tipoAjuste, cantidad), "Éxito", JOptionPane.INFORMATION\_MESSAGE);

}

// Método para crear el panel de devoluciones

private JPanel crearPanelDevoluciones() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// Panel de pestañas para los tipos de devolución

JTabbedPane tabsDevoluciones = new JTabbedPane();

tabsDevoluciones.addTab("Devolución de Cliente", crearPanelDevolucionCliente());

tabsDevoluciones.addTab("Devolución a Proveedor", crearPanelDevolucionProveedor());

tabsDevoluciones.addTab("Historial", crearPanelHistorialDevoluciones());

panel.add(tabsDevoluciones, BorderLayout.CENTER);

return panel;

}

private JPanel crearPanelDevolucionCliente() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Panel de formulario

JPanel formPanel = new JPanel(new GridLayout(6, 2, 10, 10));

// Campo ID Producto

JTextField txtIdProducto = new JTextField();

JButton btnBuscarProducto = createStyledButton("Buscar", new Color(70, 130, 180));

JPanel panelIdProducto = new JPanel(new BorderLayout(5, 5));

panelIdProducto.add(txtIdProducto, BorderLayout.CENTER);

panelIdProducto.add(btnBuscarProducto, BorderLayout.EAST);

formPanel.add(new JLabel("ID Producto:"));

formPanel.add(panelIdProducto);

// Campo Nombre Producto (solo lectura)

JTextField txtNombreProducto = new JTextField();

txtNombreProducto.setEditable(false);

formPanel.add(new JLabel("Nombre Producto:"));

formPanel.add(txtNombreProducto);

// Campo Cantidad

JSpinner spnCantidad = new JSpinner(new SpinnerNumberModel(1, 1, 10000, 1));

formPanel.add(new JLabel("Cantidad:"));

formPanel.add(spnCantidad);

// Campo Motivo

JComboBox<String> cbMotivo = new JComboBox<>(new String[]{

"Producto defectuoso",

"Producto no solicitado",

"Producto vencido",

"Producto incorrecto",

"Otro"

});

formPanel.add(new JLabel("Motivo:"));

formPanel.add(cbMotivo);

// Campo ID Transacción Original

JTextField txtIdTransaccion = new JTextField();

formPanel.add(new JLabel("ID Venta Original:"));

formPanel.add(txtIdTransaccion);

// Campo Observaciones

JTextField txtObservaciones = new JTextField();

formPanel.add(new JLabel("Observaciones:"));

formPanel.add(txtObservaciones);

// Botón Registrar

JButton btnRegistrar = createStyledButton("Registrar Devolución", new Color(63, 142, 77));

btnRegistrar.addActionListener(e -> {

if (validarDevolucion(txtIdProducto.getText(), (Integer)spnCantidad.getValue())) {

registrarDevolucionCliente(

txtIdProducto.getText(),

txtNombreProducto.getText(),

(Integer)spnCantidad.getValue(),

(String)cbMotivo.getSelectedItem(),

txtIdTransaccion.getText(),

txtObservaciones.getText()

);

}

});

// Configurar acción del botón buscar

btnBuscarProducto.addActionListener(e -> {

Producto producto = buscarProducto(txtIdProducto.getText());

if (producto != null) {

txtNombreProducto.setText(producto.getNombre());

spnCantidad.setModel(new SpinnerNumberModel(

1, 1, producto.getCantidadDisponible(), 1));

}

});

// Panel principal

panel.add(formPanel, BorderLayout.CENTER);

panel.add(btnRegistrar, BorderLayout.SOUTH);

return panel;

}

private JPanel crearPanelDevolucionProveedor() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Panel de formulario

JPanel formPanel = new JPanel(new GridLayout(6, 2, 10, 10));

// Campo ID Producto

JTextField txtIdProducto = new JTextField();

JButton btnBuscarProducto = createStyledButton("Buscar", new Color(70, 130, 180));

JPanel panelIdProducto = new JPanel(new BorderLayout(5, 5));

panelIdProducto.add(txtIdProducto, BorderLayout.CENTER);

panelIdProducto.add(btnBuscarProducto, BorderLayout.EAST);

formPanel.add(new JLabel("ID Producto:"));

formPanel.add(panelIdProducto);

// Campo Nombre Producto (solo lectura)

JTextField txtNombreProducto = new JTextField();

txtNombreProducto.setEditable(false);

formPanel.add(new JLabel("Nombre Producto:"));

formPanel.add(txtNombreProducto);

// Campo Cantidad

JSpinner spnCantidad = new JSpinner(new SpinnerNumberModel(1, 1, 10000, 1));

formPanel.add(new JLabel("Cantidad:"));

formPanel.add(spnCantidad);

// Campo Motivo

JComboBox<String> cbMotivo = new JComboBox<>(new String[]{

"Producto defectuoso",

"Producto vencido",

"Producto incorrecto",

"Exceso de entrega",

"Otro"

});

formPanel.add(new JLabel("Motivo:"));

formPanel.add(cbMotivo);

// Campo ID Transacción Original

JTextField txtIdTransaccion = new JTextField();

formPanel.add(new JLabel("ID Compra Original:"));

formPanel.add(txtIdTransaccion);

// Campo Observaciones

JTextField txtObservaciones = new JTextField();

formPanel.add(new JLabel("Observaciones:"));

formPanel.add(txtObservaciones);

// Botón Registrar

JButton btnRegistrar = createStyledButton("Registrar Devolución", new Color(192, 57, 43));

btnRegistrar.addActionListener(e -> {

if (validarDevolucion(txtIdProducto.getText(), (Integer)spnCantidad.getValue())) {

registrarDevolucionProveedor(

txtIdProducto.getText(),

txtNombreProducto.getText(),

(Integer)spnCantidad.getValue(),

(String)cbMotivo.getSelectedItem(),

txtIdTransaccion.getText(),

txtObservaciones.getText()

);

}

});

// Configurar acción del botón buscar

btnBuscarProducto.addActionListener(e -> {

Producto producto = buscarProducto(txtIdProducto.getText());

if (producto != null) {

txtNombreProducto.setText(producto.getNombre());

spnCantidad.setModel(new SpinnerNumberModel(

1, 1, producto.getCantidadDisponible(), 1));

}

});

// Panel principal

panel.add(formPanel, BorderLayout.CENTER);

panel.add(btnRegistrar, BorderLayout.SOUTH);

return panel;

}

private JPanel crearPanelHistorialDevoluciones() {

JPanel panel = new JPanel(new BorderLayout(10, 10));

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

// Modelo de tabla para las devoluciones

String[] columnas = {"ID", "Producto", "Cantidad", "Tipo", "Motivo", "Fecha", "Estado"};

Object[][] datos = obtenerDatosDevoluciones();

JTable tabla = new JTable(datos, columnas);

JScrollPane scrollPane = new JScrollPane(tabla);

// Panel de botones

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 10));

// Botón para actualizar

JButton btnActualizar = createStyledButton("Actualizar", new Color(70, 130, 180));

btnActualizar.addActionListener(e -> actualizarTablaDevoluciones(tabla));

// Botón para generar reporte

JButton btnReporte = createStyledButton("Generar Reporte", new Color(63, 142, 77));

btnReporte.addActionListener(e -> generarReporteDevoluciones());

buttonPanel.add(btnActualizar);

buttonPanel.add(btnReporte);

panel.add(scrollPane, BorderLayout.CENTER);

panel.add(buttonPanel, BorderLayout.SOUTH);

return panel;

}

private Object[][] obtenerDatosDevoluciones() {

List<Devolucion> devoluciones = controlador.obtenerDevoluciones();

Object[][] datos = new Object[devoluciones.size()][7];

for (int i = 0; i < devoluciones.size(); i++) {

Devolucion d = devoluciones.get(i);

datos[i][0] = d.getId();

datos[i][1] = d.getNombreProducto();

datos[i][2] = d.getCantidad();

datos[i][3] = d.getTipo();

datos[i][4] = d.getMotivo();

datos[i][5] = new SimpleDateFormat("dd/MM/yyyy").format(d.getFecha());

datos[i][6] = d.getEstado();

}

return datos;

}

private void actualizarTablaDevoluciones(JTable tabla) {

Object[][] nuevosDatos = obtenerDatosDevoluciones();

DefaultTableModel model = new DefaultTableModel(nuevosDatos,

new String[]{"ID", "Producto", "Cantidad", "Tipo", "Motivo", "Fecha", "Estado"});

tabla.setModel(model);

}

private boolean validarDevolucion(String idProducto, int cantidad) {

if (idProducto == null || idProducto.trim().isEmpty()) {

JOptionPane.showMessageDialog(this, "Ingrese un ID de producto", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

if (cantidad <= 0) {

JOptionPane.showMessageDialog(this, "La cantidad debe ser mayor a cero", "Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

private void registrarDevolucionCliente(String idProducto, String nombreProducto, int cantidad,

String motivo, String idTransaccion, String observaciones) {

Devolucion devolucion = new Devolucion(

"DEV-" + System.currentTimeMillis(),

idProducto,

nombreProducto,

cantidad,

"CLIENTE",

motivo,

new Date(),

"PENDIENTE",

observaciones,

idTransaccion,

usuario.getUsername()

);

if (controlador.registrarDevolucion(devolucion)) {

JOptionPane.showMessageDialog(this,

"Devolución de cliente registrada exitosamente",

"Éxito", JOptionPane.INFORMATION\_MESSAGE);

} else {

JOptionPane.showMessageDialog(this,

"Error al registrar la devolución",

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void registrarDevolucionProveedor(String idProducto, String nombreProducto, int cantidad,

String motivo, String idTransaccion, String observaciones) {

Devolucion devolucion = new Devolucion(

"DEV-" + System.currentTimeMillis(),

idProducto,

nombreProducto,

cantidad,

"PROVEEDOR",

motivo,

new Date(),

"PENDIENTE",

observaciones,

idTransaccion,

usuario.getUsername()

);

if (controlador.registrarDevolucion(devolucion)) {

JOptionPane.showMessageDialog(this,

"Devolución a proveedor registrada exitosamente",

"Éxito", JOptionPane.INFORMATION\_MESSAGE);

} else {

JOptionPane.showMessageDialog(this,

"Error al registrar la devolución",

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void generarReporteDevoluciones() {

JDialog reporteDialog = new JDialog(this, "Generar Reporte de Devoluciones", true);

reporteDialog.setSize(400, 300);

reporteDialog.setLocationRelativeTo(this);

reporteDialog.setLayout(new GridLayout(0, 1, 10, 10));

reporteDialog.getContentPane().setBackground(new Color(240, 245, 250));

// Filtros

JPanel filtroPanel = new JPanel(new GridLayout(3, 2, 5, 5));

filtroPanel.setBorder(BorderFactory.createTitledBorder("Filtros"));

// Tipo de devolución

JComboBox<String> cbTipo = new JComboBox<>(new String[]{"Todas", "CLIENTE", "PROVEEDOR"});

filtroPanel.add(new JLabel("Tipo:"));

filtroPanel.add(cbTipo);

// Estado

JComboBox<String> cbEstado = new JComboBox<>(new String[]{"Todos", "PENDIENTE", "PROCESADA", "RECHAZADA"});

filtroPanel.add(new JLabel("Estado:"));

filtroPanel.add(cbEstado);

// Rango de fechas

JSpinner spnFechaDesde = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor fechaEditorDesde = new JSpinner.DateEditor(spnFechaDesde, "dd/MM/yyyy");

spnFechaDesde.setEditor(fechaEditorDesde);

JSpinner spnFechaHasta = new JSpinner(new SpinnerDateModel());

JSpinner.DateEditor fechaEditorHasta = new JSpinner.DateEditor(spnFechaHasta, "dd/MM/yyyy");

spnFechaHasta.setEditor(fechaEditorHasta);

JPanel fechaPanel = new JPanel(new GridLayout(1, 2, 5, 5));

fechaPanel.add(spnFechaDesde);

fechaPanel.add(spnFechaHasta);

filtroPanel.add(new JLabel("Rango Fechas:"));

filtroPanel.add(fechaPanel);

// Formatos de reporte

JPanel formatoPanel = new JPanel(new GridLayout(2, 2, 5, 5));

formatoPanel.setBorder(BorderFactory.createTitledBorder("Formato"));

JRadioButton rbPDF = new JRadioButton("PDF", true);

JRadioButton rbExcel = new JRadioButton("Excel");

JRadioButton rbCSV = new JRadioButton("CSV");

JRadioButton rbHTML = new JRadioButton("HTML");

ButtonGroup bgFormato = new ButtonGroup();

bgFormato.add(rbPDF);

bgFormato.add(rbExcel);

bgFormato.add(rbCSV);

bgFormato.add(rbHTML);

formatoPanel.add(rbPDF);

formatoPanel.add(rbExcel);

formatoPanel.add(rbCSV);

formatoPanel.add(rbHTML);

// Botones

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));

JButton btnGenerar = createStyledButton("Generar Reporte", new Color(70, 130, 180));

btnGenerar.addActionListener(e -> {

String tipo = (String)cbTipo.getSelectedItem();

String estado = (String)cbEstado.getSelectedItem();

Date desde = (Date)spnFechaDesde.getValue();

Date hasta = (Date)spnFechaHasta.getValue();

String formato = "";

if (rbPDF.isSelected()) formato = "PDF";

else if (rbExcel.isSelected()) formato = "Excel";

else if (rbCSV.isSelected()) formato = "CSV";

else if (rbHTML.isSelected()) formato = "HTML";

generarReporte(tipo, estado, desde, hasta, formato);

reporteDialog.dispose();

});

JButton btnCancelar = createStyledButton("Cancelar", new Color(192, 57, 43));

btnCancelar.addActionListener(e -> reporteDialog.dispose());

buttonPanel.add(btnGenerar);

buttonPanel.add(btnCancelar);

reporteDialog.add(filtroPanel);

reporteDialog.add(formatoPanel);

reporteDialog.add(buttonPanel);

reporteDialog.setVisible(true);

}

private void generarReporte(String tipo, String estado, Date desde, Date hasta, String formato) {

// Filtrar las devoluciones según los parámetros

List<Devolucion> devoluciones = controlador.obtenerDevoluciones();

List<Devolucion> devolucionesFiltradas = new ArrayList<>();

for (Devolucion d : devoluciones) {

// Filtrar por tipo

if (!tipo.equals("Todas") && !d.getTipo().equals(tipo)) {

continue;

}

// Filtrar por estado

if (!estado.equals("Todos") && !d.getEstado().equals(estado)) {

continue;

}

// Filtrar por fecha

if (desde != null && hasta != null &&

(d.getFecha().before(desde) || d.getFecha().after(hasta))) {

continue;

}

devolucionesFiltradas.add(d);

}

// Generar el reporte según el formato seleccionado

switch (formato) {

case "PDF":

generarReportePDF(devolucionesFiltradas);

break;

case "Excel":

generarReporteExcel(devolucionesFiltradas);

break;

case "CSV":

generarReporteCSV(devolucionesFiltradas);

break;

case "HTML":

generarReporteHTML(devolucionesFiltradas);

break;

}

JOptionPane.showMessageDialog(this,

"Reporte generado exitosamente en formato " + formato,

"Éxito", JOptionPane.INFORMATION\_MESSAGE);

}

private void generarReportePDF(List<Devolucion> devoluciones) {

// Implementación para generar PDF (usando librería como iText)

try {

// Crear directorio de reportes si no existe

File dir = new File("reportes");

if (!dir.exists()) {

dir.mkdir();

}

String ruta = "reportes/devoluciones\_" + new SimpleDateFormat("yyyyMMddHHmmss").format(new Date()) + ".pdf";

// Crear documento PDF

Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream(ruta));

document.open();

// Título

com.itextpdf.text.Font fontTitulo = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 18);

Paragraph titulo = new Paragraph("Reporte de Devoluciones", fontTitulo);

titulo.setAlignment(Element.ALIGN\_CENTER);

document.add(titulo);

// Fecha de generación

com.itextpdf.text.Font fontFecha = FontFactory.getFont(FontFactory.HELVETICA, 10);

Paragraph fecha = new Paragraph("Generado el: " + new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new Date()), fontFecha);

fecha.setAlignment(Element.ALIGN\_CENTER);

document.add(fecha);

document.add(new Paragraph(" ")); // Espacio

// Crear tabla

PdfPTable table = new PdfPTable(7); // 7 columnas

table.setWidthPercentage(100);

// Encabezados de tabla

String[] headers = {"ID", "Producto", "Cantidad", "Tipo", "Motivo", "Fecha", "Estado"};

for (String header : headers) {

PdfPCell cell = new PdfPCell(new Phrase(header));

cell.setBackgroundColor(BaseColor.LIGHT\_GRAY);

table.addCell(cell);

}

// Datos

for (Devolucion d : devoluciones) {

table.addCell(d.getId());

table.addCell(d.getNombreProducto());

table.addCell(String.valueOf(d.getCantidad()));

table.addCell(d.getTipo());

table.addCell(d.getMotivo());

table.addCell(new SimpleDateFormat("dd/MM/yyyy").format(d.getFecha()));

table.addCell(d.getEstado());

}

document.add(table);

document.close();

// Abrir el archivo generado

Desktop.getDesktop().open(new File(ruta));

} catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Error al generar PDF: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void generarReporteExcel(List<Devolucion> devoluciones) {

// Implementación para generar Excel (usando librería como Apache POI)

try {

// Crear directorio de reportes si no existe

File dir = new File("reportes");

if (!dir.exists()) {

dir.mkdir();

}

String ruta = "reportes/devoluciones\_" + new SimpleDateFormat("yyyyMMddHHmmss").format(new Date()) + ".xlsx";

XSSFWorkbook workbook = new XSSFWorkbook();

Sheet sheet = workbook.createSheet("Devoluciones");

// Crear fila de encabezados

Row headerRow = sheet.createRow(0);

String[] headers = {"ID", "Producto", "Cantidad", "Tipo", "Motivo", "Fecha", "Estado"};

for (int i = 0; i < headers.length; i++) {

Cell cell = headerRow.createCell(i);

cell.setCellValue(headers[i]);

}

// Llenar datos

int rowNum = 1;

for (Devolucion d : devoluciones) {

Row row = sheet.createRow(rowNum++);

row.createCell(0).setCellValue(d.getId());

row.createCell(1).setCellValue(d.getNombreProducto());

row.createCell(2).setCellValue(d.getCantidad());

row.createCell(3).setCellValue(d.getTipo());

row.createCell(4).setCellValue(d.getMotivo());

row.createCell(5).setCellValue(new SimpleDateFormat("dd/MM/yyyy").format(d.getFecha()));

row.createCell(6).setCellValue(d.getEstado());

}

// Autoajustar columnas

for (int i = 0; i < headers.length; i++) {

sheet.autoSizeColumn(i);

}

// Escribir archivo

FileOutputStream outputStream = new FileOutputStream(ruta);

workbook.write(outputStream);

workbook.close();

outputStream.close();

// Abrir el archivo generado

Desktop.getDesktop().open(new File(ruta));

} catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Error al generar Excel: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void generarReporteCSV(List<Devolucion> devoluciones) {

try {

// Crear directorio de reportes si no existe

File dir = new File("reportes");

if (!dir.exists()) {

dir.mkdir();

}

String ruta = "reportes/devoluciones\_" + new SimpleDateFormat("yyyyMMddHHmmss").format(new Date()) + ".csv";

FileWriter writer = new FileWriter(ruta);

// Escribir encabezados

writer.append("ID,Producto,Cantidad,Tipo,Motivo,Fecha,Estado\n");

// Escribir datos

for (Devolucion d : devoluciones) {

writer.append(d.getId()).append(",");

writer.append(d.getNombreProducto()).append(",");

writer.append(String.valueOf(d.getCantidad())).append(",");

writer.append(d.getTipo()).append(",");

writer.append(d.getMotivo()).append(",");

writer.append(new SimpleDateFormat("dd/MM/yyyy").format(d.getFecha())).append(",");

writer.append(d.getEstado()).append("\n");

}

writer.close();

// Abrir el archivo generado

Desktop.getDesktop().open(new File(ruta));

} catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Error al generar CSV: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void generarReporteHTML(List<Devolucion> devoluciones) {

try {

// Crear directorio de reportes si no existe

File dir = new File("reportes");

if (!dir.exists()) {

dir.mkdir();

}

String ruta = "reportes/devoluciones\_" + new SimpleDateFormat("yyyyMMddHHmmss").format(new Date()) + ".html";

FileWriter writer = new FileWriter(ruta);

// Escribir encabezado HTML

writer.write("<!DOCTYPE html>\n");

writer.write("<html>\n");

writer.write("<head>\n");

writer.write("<title>Reporte de Devoluciones</title>\n");

writer.write("<style>\n");

writer.write("body { font-family: Arial, sans-serif; }\n");

writer.write("h1 { text-align: center; color: #333; }\n");

writer.write(".fecha { text-align: center; color: #666; margin-bottom: 20px; }\n");

writer.write("table { width: 100%; border-collapse: collapse; margin-top: 20px; }\n");

writer.write("th { background-color: #f2f2f2; text-align: left; padding: 8px; }\n");

writer.write("td { padding: 8px; border-bottom: 1px solid #ddd; }\n");

writer.write("tr:nth-child(even) { background-color: #f9f9f9; }\n");

writer.write("</style>\n");

writer.write("</head>\n");

writer.write("<body>\n");

writer.write("<h1>Reporte de Devoluciones</h1>\n");

writer.write("<div class=\"fecha\">Generado el: " +

new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new Date()) + "</div>\n");

writer.write("<table>\n");

writer.write("<tr>\n");

writer.write("<th>ID</th>\n");

writer.write("<th>Producto</th>\n");

writer.write("<th>Cantidad</th>\n");

writer.write("<th>Tipo</th>\n");

writer.write("<th>Motivo</th>\n");

writer.write("<th>Fecha</th>\n");

writer.write("<th>Estado</th>\n");

writer.write("</tr>\n");

// Escribir datos

for (Devolucion d : devoluciones) {

writer.write("<tr>\n");

writer.write("<td>" + d.getId() + "</td>\n");

writer.write("<td>" + d.getNombreProducto() + "</td>\n");

writer.write("<td>" + d.getCantidad() + "</td>\n");

writer.write("<td>" + d.getTipo() + "</td>\n");

writer.write("<td>" + d.getMotivo() + "</td>\n");

writer.write("<td>" + new SimpleDateFormat("dd/MM/yyyy").format(d.getFecha()) + "</td>\n");

writer.write("<td>" + d.getEstado() + "</td>\n");

writer.write("</tr>\n");

}

writer.write("</table>\n");

writer.write("</body>\n");

writer.write("</html>\n");

writer.close();

// Abrir el archivo generado

Desktop.getDesktop().open(new File(ruta));

} catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Error al generar HTML: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

}

}

private void abrirReporteInventario() {

try {

// Crear el panel de contenido principal con CardLayout

JPanel mainContentPanel = new JPanel();

CardLayout cardLayout = new CardLayout();

mainContentPanel.setLayout(cardLayout);

// Crear el panel contenedor que tendrá el CardLayout

JPanel containerPanel = new JPanel(new BorderLayout());

containerPanel.add(mainContentPanel, BorderLayout.CENTER);

// Crear el controlador de reportes

ReportesControlador reportesControlador = new ReportesControlador(null, usuario);

// Crear el panel de reportes de inventario

ReporteInventarioPanel reportePanel = new ReporteInventarioPanel(

usuario,

reportesControlador,

cardLayout,

mainContentPanel

);

// Agregar el panel de reportes al contenedor principal

mainContentPanel.add(reportePanel, "reporteInventario");

// Crear el diálogo para mostrar el reporte

JDialog reportDialog = new JDialog(this, "Reportes de Inventario", true);

reportDialog.setSize(1000, 700);

reportDialog.setLocationRelativeTo(this);

reportDialog.setLayout(new BorderLayout());

reportDialog.add(containerPanel, BorderLayout.CENTER);

// Mostrar el panel de reportes

cardLayout.show(mainContentPanel, "reporteInventario");

reportDialog.setVisible(true);

} catch (Exception e) {

JOptionPane.showMessageDialog(this,

"Error al abrir reportes: " + e.getMessage(),

"Error", JOptionPane.ERROR\_MESSAGE);

e.printStackTrace();

}

}

}

Captura de pantalla de un celular con texto e imágenes

El contenido generado por IA puede ser incorrecto.

# Cliente

package Modelo;

import java.util.Date;

import java.util.Objects;

import Controlador.ClientesContro;

public class Cliente {

private String id;

private String telefono;

private String nombre;

private String ultimaCompra;

private int puntos;

private Date fechaRegistro;

private Date fechaEliminacion;

public Cliente() {

this.fechaRegistro = new Date(); // Fecha actual por defecto

}

// Constructor con parámetros esenciales

public Cliente(String telefono, String nombre) {

this();

this.telefono = telefono;

this.nombre = nombre;

}

// Constructor completo

public Cliente(String id, String telefono, String nombre, String ultimaCompra,

int puntos, Date fechaRegistro, Date fechaEliminacion) {

this.id = id;

this.telefono = telefono;

this.nombre = nombre;

this.ultimaCompra = ultimaCompra;

this.puntos = puntos;

this.fechaRegistro = fechaRegistro != null ? fechaRegistro : new Date();

this.fechaEliminacion = fechaEliminacion;

}

public void copiarDe(Cliente otro) {

this.id = otro.id;

this.telefono = otro.telefono;

this.nombre = otro.nombre;

this.ultimaCompra = otro.ultimaCompra;

this.puntos = otro.puntos;

this.fechaRegistro = otro.fechaRegistro;

this.fechaEliminacion = otro.fechaEliminacion;

}

public String getTelefono() {

return telefono;

}

public void setTelefono(String telefono) {

this.telefono = telefono;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public String getUltimaCompra() {

return ultimaCompra;

}

public void setUltimaCompra(String ultimaCompra) {

this.ultimaCompra = ultimaCompra;

}

public int getPuntos() {

return puntos;

}

public void setPuntos(int puntos) {

this.puntos = puntos;

}

public void setFechaRegistro(Date fechaRegistro) {

this.fechaRegistro = fechaRegistro;

}

// Getter para el campo fechaRegistro

public Date getFechaRegistro() {

return fechaRegistro;

}

public Date getFechaEliminacion() {

return fechaEliminacion; // Método getter para fechaEliminacion

}

public void setFechaEliminacion(Date fechaEliminacion) {

this.fechaEliminacion = fechaEliminacion; // Método setter para fechaEliminacion

}

public void setId(String id) {

this.id = id;

}

// Getter para el campo id

public String getId() {

return id;

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false;

Cliente other = (Cliente) obj;

return Objects.equals(id, other.id) &&

Objects.equals(telefono, other.telefono) &&

Objects.equals(nombre, other.nombre) &&

Objects.equals(ultimaCompra, other.ultimaCompra) &&

puntos == other.puntos;

}

@Override

public int hashCode() {

return Objects.hash(id, telefono, nombre, ultimaCompra, puntos);

}

}

package Controlador;

import Modelo.Cliente;

import Modelo.ClienteDAO;

import Modelo.ClienteDAOImpl;

import Vista.clientes;

import java.util.Date;

import java.util.List;

import java.util.Objects;

import javax.swing.JOptionPane;

import javax.swing.JTable;

import javax.swing.SwingUtilities;

import javax.swing.table.DefaultTableModel;

public class ClientesContro {

private final ClienteDAO modelo;

private final clientes vista;

private final Cliente cliente;

private String telefono, id;

private final DefaultTableModel modeloTabla;

public ClientesContro(ClienteDAO modelo, clientes vista, Cliente cliente, DefaultTableModel modeloTabla) {

Objects.requireNonNull(modelo, "El modelo ClienteDAO no puede ser nulo");

Objects.requireNonNull(vista, "La vista no puede ser nula");

Objects.requireNonNull(modeloTabla, "El modelo de tabla no puede ser nulo");

this.modelo = modelo;

this.vista = vista;

this.cliente = cliente;

this.modeloTabla = modeloTabla;

// Verificar y preparar la tabla al iniciar

if (modelo instanceof ClienteDAOImpl) {

((ClienteDAOImpl)modelo).verificarYEstructurarTabla();

}

inicializarControlador();

}

private void inicializarControlador() {

SwingUtilities.invokeLater(() -> {

configurarListeners();

cargarClientesIniciales();

});

}

private void configurarListeners() {

vista.getBtnAgregar().addActionListener(e -> agregarCliente());

vista.getBtnEditar().addActionListener(e -> editarCliente());

vista.getBtnEliminar().addActionListener(e -> eliminarCliente());

}

private void cargarClientesIniciales() {

try {

List<Cliente> clientes = modelo.obtenerTodos();

// Actualizar la vista con los clientes...

} catch (Exception e) {

mostrarError("Error al cargar clientes: " + e.getMessage());

}

}

private void mostrarError(String mensaje) {

JOptionPane.showMessageDialog(vista, mensaje, "Error", JOptionPane.ERROR\_MESSAGE);

}

private void cargarClientes() {

modeloTabla.setRowCount(0); // Limpiar tabla ANTES de cargar

List<Cliente> clientes = modelo.obtenerTodos();

for (Cliente cliente : clientes) {

modeloTabla.addRow(new Object[]{

cliente.getId(),

cliente.getTelefono(),

cliente.getNombre(),

cliente.getUltimaCompra(),

cliente.getPuntos()

});

}

}

public void agregarCliente() {

Cliente nuevoCliente = new Cliente();

if (!vista.mostrarFormularioCliente(nuevoCliente)) {

return; // Si se canceló

}

try {

// Generar ID único para el nuevo cliente

nuevoCliente.setId(generarNuevoId());

// Establecer fecha de registro

nuevoCliente.setFechaRegistro(new Date());

// Validar datos antes de insertar

validarDatosCliente(nuevoCliente);

// Insertar en la base de datos

modelo.agregarCliente(nuevoCliente);

// Actualizar vista

vista.actualizarTablaClientes();

actualizarVistaDespuesDeAgregar(nuevoCliente);

} catch (Exception e) {

vista.mostrarError("Error al guardar: " + e.getMessage());

e.printStackTrace();

}

}

// Métodos auxiliares

private void validarDatosCliente(Cliente cliente) {

if (cliente == null) {

throw new IllegalArgumentException("Cliente no puede ser nulo");

}

if (cliente.getTelefono() == null || cliente.getTelefono().trim().isEmpty()) {

throw new IllegalArgumentException("Teléfono es requerido");

}

if (!cliente.getTelefono().matches("\\d{7,15}")) {

throw new IllegalArgumentException("Teléfono debe contener solo números (7-15 dígitos)");

}

if (cliente.getNombre() == null || cliente.getNombre().trim().isEmpty()) {

throw new IllegalArgumentException("Nombre es requerido");

}

// Verificar si ya existe un cliente con ese teléfono

Cliente existente = modelo.buscarPorTelefono(cliente.getTelefono());

if (existente != null) {

throw new IllegalStateException("Ya existe un cliente con este teléfono");

}

}

private void actualizarVistaDespuesDeAgregar(Cliente cliente) {

cargarClientes(); // Recargar datos

// Seleccionar el nuevo cliente en la tabla

int fila = encontrarFilaCliente(cliente.getId());

if (fila >= 0) {

vista.getTablaClientes().setRowSelectionInterval(fila, fila);

vista.getTablaClientes().scrollRectToVisible(

vista.getTablaClientes().getCellRect(fila, 0, true));

}

}

private int encontrarFilaCliente(String idCliente) {

DefaultTableModel model = (DefaultTableModel) vista.getTablaClientes().getModel();

for (int i = 0; i < model.getRowCount(); i++) {

if (idCliente.equals(model.getValueAt(i, 0))) {

return i;

}

}

return -1;

}

private String generarNuevoId() {

return "CLI-" + System.currentTimeMillis();

}

public void editarCliente() {

try {

// Obtener el ID del cliente seleccionado

String id = vista.getSelectedClienteId();

if (id == null) return; // Salida silenciosa

Cliente clienteOriginal = modelo.buscarPorId(id);

if (clienteOriginal == null) return;

// Crear copia para edición

Cliente clienteEditable = new Cliente();

clienteEditable.copiarDe(clienteOriginal);

// Mostrar formulario sin validaciones previas

if (vista.mostrarFormularioEdicion(clienteEditable)) {

// Aplicar cambios directamente

if (!clienteOriginal.getTelefono().equals(clienteEditable.getTelefono())) {

if (modelo.buscarPorTelefono(clienteEditable.getTelefono()) != null) {

vista.mostrarError("El teléfono ya está registrado");

return;

}

}

// Actualizar campos

clienteOriginal.setTelefono(clienteEditable.getTelefono());

clienteOriginal.setNombre(clienteEditable.getNombre());

clienteOriginal.setUltimaCompra(clienteEditable.getUltimaCompra());

modelo.actualizarCliente(clienteOriginal);

vista.actualizarTablaClientes();

}

} catch (Exception e) {

vista.mostrarError("Error durante la edición: " + e.getMessage());

}

}

private void seleccionarClienteEnTabla(String id) {

JTable tabla = vista.getTablaClientes();

DefaultTableModel model = (DefaultTableModel) tabla.getModel();

for (int i = 0; i < model.getRowCount(); i++) {

if (id.equals(model.getValueAt(i, 0))) {

tabla.setRowSelectionInterval(i, i);

tabla.scrollRectToVisible(tabla.getCellRect(i, 0, true));

break;

}

}

}

public void eliminarCliente() {

JTable tabla = vista.getTablaClientes();

int fila = tabla.getSelectedRow();

// Si no hay fila seleccionada, simplemente retornar sin mensaje

if (fila < 0) return;

String id = tabla.getValueAt(fila, 0).toString();

Cliente cliente = modelo.buscarPorId(id);

// Si el cliente no existe, mostrar mensaje de error y retornar

if (cliente != null && vista.mostrarConfirmacion("¿Eliminar a " + cliente.getNombre() + "?")) {

modelo.eliminarCliente(id);

((DefaultTableModel)tabla.getModel()).removeRow(fila);

}

}

public void actualizarCliente(Cliente clienteActualizado) {

modelo.actualizarCliente(clienteActualizado);

cargarClientes(); // Recargar datos

}

public Cliente buscarClientePorTelefono(String telefono) {

if (telefono == null || telefono.trim().isEmpty()) {

throw new IllegalArgumentException("El teléfono no puede estar vacío");

}

try {

return modelo.buscarPorTelefono(telefono);

} catch (Exception e) {

// Loggear el error

System.err.println("Error al buscar cliente por teléfono: " + e.getMessage());

throw new RuntimeException("Error al buscar cliente", e);

}

}

public List<Cliente> obtenerTodosClientes() {

try {

List<Cliente> todosClientes = modelo.obtenerTodos();

// Filtrar clientes eliminados

// return todosClientes.stream()

// .filter(c -> c.getFechaEliminacion() == null)

// .collect(Collectors.toList());

return todosClientes;

} catch (Exception e) {

// Loggear el error

System.err.println("Error al obtener todos los clientes: " + e.getMessage());

throw new RuntimeException("Error al obtener clientes", e);

}

}

public Cliente buscarClientePorId(String id) {

try {

// Verificar que el ID no sea nulo o vacío

if (id == null || id.trim().isEmpty()) {

vista.mostrarError("El ID no puede estar vacío");

return null;

}

// Buscar en el modelo usando el ID

Cliente cliente = modelo.buscarPorId(id);

if (cliente == null) {

vista.mostrarError("No se encontró cliente con ID: " + id);

}

return cliente;

} catch (Exception e) {

vista.mostrarError("Error al buscar cliente por ID: " + e.getMessage());

return null;

}

}

}

package Modelo;

import java.sql.Connection;

import java.util.List;

import ConexionBD.ConexionAccess;

public interface ClienteDAO {

List<Cliente> obtenerTodos();

Cliente buscarPorId(String id);

Cliente buscarPorTelefono(String telefono);

void agregarCliente(Cliente cliente);

void actualizarCliente(Cliente cliente);

void eliminarCliente(String id);

Connection conn = ConexionAccess.conectar();

}

package Modelo;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import javax.swing.JOptionPane;

import ConexionBD.ConexionAccess;

public class ClienteDAOImpl implements ClienteDAO {

@Override

public List<Cliente> obtenerTodos() {

List<Cliente> clientes = new ArrayList<>();

String query = "SELECT \* FROM clientes WHERE fecha\_eliminacion IS NULL";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query)) {

while (rs.next()) {

Cliente cliente = new Cliente(

rs.getString("id"),

rs.getString("telefono"),

rs.getString("nombre"),

rs.getString("ultima\_compra"),

rs.getInt("puntos"),

rs.getDate("fecha\_registro"),

rs.getDate("fecha\_eliminacion")

);

clientes.add(cliente);

}

} catch (SQLException e) {

System.err.println("⚠️ Error al cargar clientes: " + e.getMessage());

}

return clientes;

}

@Override

public Cliente buscarPorId(String id) {

String query = "SELECT \* FROM clientes WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setString(1, id);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return new Cliente(

rs.getString("id"),

rs.getString("telefono"),

rs.getString("nombre"),

rs.getString("ultima\_compra"),

rs.getInt("puntos"),

rs.getDate("fecha\_registro"),

rs.getDate("fecha\_eliminacion")

);

}

} catch (SQLException e) {

System.err.println("⚠️ Error al buscar cliente por ID: " + e.getMessage());

}

return null;

}

@Override

public Cliente buscarPorTelefono(String telefono) {

String query = "SELECT \* FROM clientes WHERE telefono = ? AND fecha\_eliminacion IS NULL";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setString(1, telefono);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return new Cliente(

rs.getString("id"),

rs.getString("telefono"),

rs.getString("nombre"),

rs.getString("ultima\_compra"),

rs.getInt("puntos"),

rs.getDate("fecha\_registro"),

rs.getDate("fecha\_eliminacion")

);

}

} catch (SQLException e) {

System.err.println("⚠️ Error al buscar cliente por teléfono: " + e.getMessage());

}

return null;

}

@Override

public void agregarCliente(Cliente cliente) {

String sql = "INSERT INTO clientes (id, telefono, nombre, ultima\_compra, puntos, fecha\_registro) VALUES (?, ?, ?, ?, ?, ?)";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

// Usar Timestamp para obtener la fecha y hora actual

java.sql.Timestamp fechaRegistro = new java.sql.Timestamp(cliente.getFechaRegistro().getTime());

// Manejar fecha de última compra (puede ser null)

java.sql.Date ultimaCompra = null;

if (cliente.getUltimaCompra() != null && !cliente.getUltimaCompra().isEmpty()) {

ultimaCompra = java.sql.Date.valueOf(cliente.getUltimaCompra());

}

pstmt.setString(1, cliente.getId());

pstmt.setString(2, cliente.getTelefono());

pstmt.setString(3, cliente.getNombre());

pstmt.setDate(4, ultimaCompra);

pstmt.setInt(5, cliente.getPuntos());

pstmt.setTimestamp(6, fechaRegistro); // Establecer Timestamp con fecha y hora

int rowsAffected = pstmt.executeUpdate();

// Solo mostrar mensaje si no se agregó correctamente

if (rowsAffected > 0) {

// mensaje de éxito, usa:

// JOptionPane.showMessageDialog(null, "Cliente agregado exitosamente");

}

} catch (Exception e) {

e.printStackTrace(); // Imprimir el error en consola

JOptionPane.showMessageDialog(null, "Error al guardar: " + e.getMessage()); // Solo mostrar en caso de error

}

}

@Override

public void actualizarCliente(Cliente cliente) {

String query = "UPDATE clientes SET telefono = ?, nombre = ?, ultima\_compra = ?, puntos = ? WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setString(1, cliente.getTelefono());

pstmt.setString(2, cliente.getNombre());

pstmt.setString(3, cliente.getUltimaCompra());

pstmt.setInt(4, cliente.getPuntos());

pstmt.setString(5, cliente.getId());

pstmt.executeUpdate();

System.out.println("✅ Cliente actualizado.");

} catch (SQLException e) {

System.err.println("❌ Error al actualizar cliente: " + e.getMessage());

throw new RuntimeException("Error en la base de datos", e);

}

}

@Override

public void eliminarCliente(String id) {

String query = "UPDATE clientes SET fecha\_eliminacion = ? WHERE id = ?"; // Borrado lógico

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setDate(1, new java.sql.Date(System.currentTimeMillis()));

pstmt.setString(2, id);

pstmt.executeUpdate();

System.out.println("✅ Cliente marcado como eliminado.");

} catch (SQLException e) {

System.err.println("❌ Error al eliminar cliente: " + e.getMessage());

throw new RuntimeException("Error en la base de datos", e);

}

}

public void verificarYEstructurarTabla() {

try (Connection conn = ConexionAccess.conectar()) {

// Verificar si la tabla existe

DatabaseMetaData meta = conn.getMetaData();

ResultSet tables = meta.getTables(null, null, "CLIENTES", new String[] {"TABLE"});

if (!tables.next()) {

// La tabla no existe, crearla

crearTablaClientes(conn);

} else {

// La tabla existe, verificar estructura

verificarEstructuraTabla(conn);

}

} catch (SQLException e) {

System.err.println("Error al verificar tabla CLIENTES: " + e.getMessage());

throw new RuntimeException("Error crítico con la tabla CLIENTES", e);

}

}

private void crearTablaClientes(Connection conn) throws SQLException {

String sql = "CREATE TABLE CLIENTES (" +

"id VARCHAR(50) PRIMARY KEY, " +

"telefono VARCHAR(15) NOT NULL, " +

"nombre VARCHAR(100) NOT NULL, " +

"ultima\_compra VARCHAR(20), " +

"puntos INTEGER DEFAULT 0, " +

"fecha\_registro DATETIME NOT NULL, " +

"fecha\_eliminacion DATETIME)";

try (Statement stmt = conn.createStatement()) {

stmt.execute(sql);

System.out.println("✅ Tabla CLIENTES creada exitosamente");

}

}

private void verificarEstructuraTabla(Connection conn) throws SQLException {

// Verificar que las columnas necesarias existan

String[] columnasRequeridas = {"id", "telefono", "nombre", "ultima\_compra", "puntos", "fecha\_registro", "fecha\_eliminacion"};

DatabaseMetaData meta = conn.getMetaData();

ResultSet columnas = meta.getColumns(null, null, "CLIENTES", null);

List<String> columnasExistentes = new ArrayList<>();

while (columnas.next()) {

columnasExistentes.add(columnas.getString("COLUMN\_NAME").toLowerCase());

}

for (String columna : columnasRequeridas) {

if (!columnasExistentes.contains(columna.toLowerCase())) {

throw new SQLException("La columna " + columna + " no existe en la tabla CLIENTES");

}

}

}

}

package Vista;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.image.BufferedImage;

import java.io.File;

import javax.imageio.ImageIO;

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import javax.swing.filechooser.FileNameExtensionFilter;

import Controlador.ReportesControlador;

import Modelo.Usuario;

import Modelo.UsuarioDAO;

public class movimientocliente extends JFrame {

private static final long serialVersionUID = 1L;

private JPanel contentPane;

private static String usuarioActual;

private JTextField idField, nombreField, telefonoField, fechaField, direccionField, emailField;

private Usuario usuario;

private String rolUsuario;

public movimientocliente (Usuario usuario) {

this.usuario = usuario;

UsuarioDAO usuarioDAO = new UsuarioDAO();

this.rolUsuario = usuario.getRol();

initUI();

this.addWindowListener(new java.awt.event.WindowAdapter() {

@Override

public void windowClosing(java.awt.event.WindowEvent windowEvent) {

menuprincipal menu = new menuprincipal(usuario);

menu.setVisible(true);

}

});

getRootPane().getInputMap(JComponent.WHEN\_IN\_FOCUSED\_WINDOW).put(

KeyStroke.getKeyStroke(KeyEvent.VK\_ESCAPE, 0), "cancelar");

getRootPane().getActionMap().put("cancelar", new AbstractAction() {

@Override

public void actionPerformed(ActionEvent e) {

dispose();

new menuprincipal(usuario).setVisible(true);

}

});

}

private void initUI() {

setTitle("El Habanerito - Gestión de Cliente");

setSize(1517, 903);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

setResizable(true);

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));

JPanel northContainer = new JPanel();

northContainer.setLayout(new BoxLayout(northContainer, BoxLayout.Y\_AXIS));

northContainer.add(crearPanelSuperior());

northContainer.add(crearMenuHorizontal());

northContainer.add(Box.createRigidArea(new Dimension(0, 10)));

mainPanel.add(northContainer, BorderLayout.NORTH);

// Panel central con el formulario

JPanel centerPanel = new JPanel(new BorderLayout());

centerPanel.setBorder(BorderFactory.createEmptyBorder(20, 50, 20, 50));

// Panel gris para el formulario

JPanel formularioPanel = new JPanel();

formularioPanel.setBackground(new Color(230, 230, 230));

formularioPanel.setLayout(new BoxLayout(formularioPanel, BoxLayout.Y\_AXIS));

formularioPanel.setBorder(BorderFactory.createEmptyBorder(30, 30, 30, 30));

// Título de la acción DENTRO del panel gris

JLabel tituloAccion = new JLabel("Información del Cliente", SwingConstants.CENTER);

tituloAccion.setFont(new Font("Arial", Font.BOLD, 24));

tituloAccion.setBorder(BorderFactory.createEmptyBorder(0, 0, 30, 0));

formularioPanel.add(tituloAccion);

// Campos del formulario - Primera fila

JPanel primeraFila = new JPanel(new FlowLayout(FlowLayout.CENTER, 50, 10));

primeraFila.setBackground(new Color(230, 230, 230));

idField = crearCampoFormulario("ID Cliente:", 150);

nombreField = crearCampoFormulario("Nombre Completo:", 250);

primeraFila.add(crearGrupoCampo(idField, "ID Cliente:"));

primeraFila.add(crearGrupoCampo(nombreField, "Nombre Completo:"));

formularioPanel.add(primeraFila);

// Campos del formulario - Segunda fila

JPanel segundaFila = new JPanel(new FlowLayout(FlowLayout.CENTER, 50, 10));

segundaFila.setBackground(new Color(230, 230, 230));

telefonoField = crearCampoFormulario("Teléfono:", 150);

fechaField = crearCampoFormulario("Fecha Registro:", 150);

segundaFila.add(crearGrupoCampo(telefonoField, "Teléfono:"));

segundaFila.add(crearGrupoCampo(fechaField, "Fecha Registro:"));

formularioPanel.add(segundaFila);

// Campos del formulario - Tercera fila

JPanel terceraFila = new JPanel(new FlowLayout(FlowLayout.CENTER, 50, 10));

terceraFila.setBackground(new Color(230, 230, 230));

direccionField = crearCampoFormulario("Dirección:", 300);

emailField = crearCampoFormulario("Correo Electrónico:", 250);

terceraFila.add(crearGrupoCampo(direccionField, "Dirección:"));

terceraFila.add(crearGrupoCampo(emailField, "Correo Electrónico:"));

formularioPanel.add(terceraFila);

// Espacio adicional antes de los botones

formularioPanel.add(Box.createRigidArea(new Dimension(0, 30)));

centerPanel.add(formularioPanel, BorderLayout.CENTER);

mainPanel.add(centerPanel, BorderLayout.CENTER);

mainPanel.add(crearPanelInferior(), BorderLayout.SOUTH);

getContentPane().add(mainPanel);

}

private JPanel crearGrupoCampo(JTextField textField, String etiqueta) {

JPanel grupo = new JPanel();

grupo.setLayout(new BoxLayout(grupo, BoxLayout.Y\_AXIS));

grupo.setBackground(new Color(230, 230, 230));

grupo.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

JLabel label = new JLabel(etiqueta);

label.setFont(new Font("Arial", Font.BOLD, 16));

label.setAlignmentX(Component.LEFT\_ALIGNMENT);

textField.setAlignmentX(Component.LEFT\_ALIGNMENT);

textField.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 30));

grupo.add(label);

grupo.add(Box.createRigidArea(new Dimension(0, 5)));

grupo.add(textField);

return grupo;

}

private JTextField crearCampoFormulario(String etiqueta, int ancho) {

JTextField textField = new JTextField();

textField.setFont(new Font("Arial", Font.PLAIN, 16));

textField.setPreferredSize(new Dimension(ancho, 30));

textField.setMaximumSize(new Dimension(ancho, 30));

return textField;

}

private JPanel crearPanelSuperior() {

JPanel topPanel = new JPanel(new BorderLayout());

topPanel.setBackground(new Color(255, 198, 144));

topPanel.setPreferredSize(new Dimension(getWidth(), 60));

topPanel.setBorder(BorderFactory.createEmptyBorder(5, 15, 5, 15));

JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 0));

rightPanel.setOpaque(false);

JButton usuarioBtn = new JButton(usuarioActual);

usuarioBtn.setFont(new Font("Arial", Font.BOLD, 14));

usuarioBtn.setForeground(Color.BLACK);

usuarioBtn.setContentAreaFilled(false);

usuarioBtn.setBorderPainted(false);

usuarioBtn.setFocusPainted(false);

usuarioBtn.setCursor(new Cursor(Cursor.HAND\_CURSOR));

usuarioBtn.addActionListener(e -> cambiarUsuario());

rightPanel.add(usuarioBtn);

try {

ImageIcon originalIcon = new ImageIcon("C:\\Users\\Anahi\\eclipse-workspace\\Punto\_Venta\\Imagenes\\logo.png");

Image originalImage = originalIcon.getImage();

int logoHeight = 60;

int logoWidth = (int) ((double) originalIcon.getIconWidth() / originalIcon.getIconHeight() \* logoHeight);

Image resizedImage = originalImage.getScaledInstance(logoWidth, logoHeight, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage));

rightPanel.add(logo, 0);

} catch (Exception e) {

System.err.println("Error cargando el logo: " + e.getMessage());

}

topPanel.add(rightPanel, BorderLayout.EAST);

return topPanel;

}

private JPanel crearMenuHorizontal() {

JPanel menuPanel = new JPanel(new GridLayout(1, 5));

menuPanel.setBackground(new Color(230, 230, 230));

menuPanel.setPreferredSize(new Dimension(getWidth(), 50));

menuPanel.setBorder(BorderFactory.createMatteBorder(1, 0, 1, 0, Color.GRAY));

String[] opciones = {"Productos", "Reportes", "Inventario", "Cliente", "Salir"};

for (String opcion : opciones) {

JButton btn = crearBotonMenu(opcion);

btn.addActionListener(e -> manejarAccionMenu(opcion));

menuPanel.add(btn);

}

return menuPanel;

}

private JButton crearBotonMenu(String texto) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

boton.setPreferredSize(new Dimension(0, 50));

boton.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 50));

boton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseEntered(java.awt.event.MouseEvent evt) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

public void mouseExited(java.awt.event.MouseEvent evt) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

});

return boton;

}

private void manejarAccionMenu(String opcion) {

switch (opcion) {

case "Salir":

this.dispose();

new menuprincipal(usuario).setVisible(true);

break;

case "Productos":

new producto(usuario).setVisible(true);

case "Reportes":

this.dispose(); // Cierra la ventana actual

reportes vistaReportes = new reportes(usuario, new ReportesControlador(null, usuario));

vistaReportes.setVisible(true); // Muestra la ventana

break;

case "Inventario":

new inventario(usuario).setVisible(true);

break;

case "Cliente":

new clientes (usuario, null).setVisible(true);

break;

}

}

private void cambiarUsuario() {

JDialog changeUserDialog = new JDialog(this, "Cambiar Usuario", true);

changeUserDialog.setSize(300, 150);

changeUserDialog.setLocationRelativeTo(this);

JPanel dialogPanel = new JPanel(new BorderLayout(10, 10));

dialogPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

JLabel instructionLabel = new JLabel("¿Desea cambiar de usuario?", SwingConstants.CENTER);

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 0));

JButton cambiarBtn = new JButton("Cambiar de Usuario");

JButton cancelarBtn = new JButton("Cancelar");

cambiarBtn.addActionListener(e -> {

changeUserDialog.dispose();

this.dispose();

new Login().setVisible(true);

});

cancelarBtn.addActionListener(e -> changeUserDialog.dispose());

buttonPanel.add(cambiarBtn);

buttonPanel.add(cancelarBtn);

dialogPanel.add(instructionLabel, BorderLayout.CENTER);

dialogPanel.add(buttonPanel, BorderLayout.SOUTH);

changeUserDialog.getContentPane().add(dialogPanel);

changeUserDialog.setVisible(true);

}

private JPanel crearPanelInferior() {

JPanel panelInferior = new JPanel(new BorderLayout());

panelInferior.setBackground(Color.PINK);

panelInferior.setBorder(BorderFactory.createEmptyBorder(10, 15, 10, 15));

panelInferior.setPreferredSize(new Dimension(getWidth(), 90));

JPanel botones = crearPanelBotonesAccion();

botones.setOpaque(false); // para que el fondo rosa se vea

panelInferior.add(botones, BorderLayout.CENTER);

return panelInferior;

}

private JPanel crearPanelBotonesAccion() {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 10));

panel.setOpaque(false);

// Botones específicos para gestión de clientes

String[] nombres = {"Editar", "Eliminar", "Agregar"};

for (String texto : nombres) {

JButton btn = new JButton(texto) {

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(getBackground());

g2.fillRoundRect(0, 0, getWidth(), getHeight(), 30, 30);

super.paintComponent(g2);

g2.dispose();

}

@Override

protected void paintBorder(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(getBackground());

g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 30, 30);

g2.dispose();

}

};

btn.setFont(new Font("Arial", Font.BOLD, 20));

btn.setBackground(Color.LIGHT\_GRAY);

btn.setForeground(Color.BLACK);

btn.setFocusPainted(false);

btn.setContentAreaFilled(false);

btn.setOpaque(false);

btn.setBorder(BorderFactory.createEmptyBorder(10, 50, 10, 50));

// Asignar acciones específicas para clientes

if (texto.equals("Guardar Cliente")) {

btn.addActionListener(e -> guardarCliente());

} else if (texto.equals("Limpiar Campos")) {

btn.addActionListener(e -> limpiarCampos());

} else if (texto.equals("Eliminar Cliente")) {

btn.addActionListener(e -> eliminarCliente());

}

panel.add(btn);

}

return panel;

}

private void guardarCliente() {

if (validarCamposCliente()) {

// Lógica para guardar cliente en base de datos

String mensaje = "Cliente guardado exitosamente:\n" +

"ID: " + idField.getText() + "\n" +

"Nombre: " + nombreField.getText() + "\n" +

"Teléfono: " + telefonoField.getText() + "\n" +

"Dirección: " + direccionField.getText() + "\n" +

"Email: " + emailField.getText();

JOptionPane.showMessageDialog(this, mensaje, "Cliente Guardado", JOptionPane.INFORMATION\_MESSAGE);

limpiarCampos();

}

}

private boolean validarCamposCliente() {

if (idField.getText().isEmpty() || nombreField.getText().isEmpty() ||

telefonoField.getText().isEmpty() || direccionField.getText().isEmpty()) {

JOptionPane.showMessageDialog(this,

"Los campos ID, Nombre, Teléfono y Dirección son obligatorios",

"Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

// Validar formato de email si está presente

if (!emailField.getText().isEmpty() && !emailField.getText().matches("^[\\w-.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {

JOptionPane.showMessageDialog(this,

"El formato del correo electrónico no es válido",

"Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

// Validar que el teléfono solo contenga números

if (!telefonoField.getText().matches("\\d+")) {

JOptionPane.showMessageDialog(this,

"El teléfono solo debe contener números",

"Error", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

private void limpiarCampos() {

idField.setText("");

nombreField.setText("");

telefonoField.setText("");

fechaField.setText("");

direccionField.setText("");

emailField.setText("");

}

private void eliminarCliente() {

if (idField.getText().isEmpty()) {

JOptionPane.showMessageDialog(this,

"Debe ingresar el ID del cliente a eliminar",

"Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int confirm = JOptionPane.showConfirmDialog(this,

"¿Está seguro que desea eliminar al cliente con ID: " + idField.getText() + "?",

"Confirmar Eliminación", JOptionPane.YES\_NO\_OPTION);

if (confirm == JOptionPane.YES\_OPTION) {

// Lógica para eliminar cliente de la base de datos

JOptionPane.showMessageDialog(this,

"Cliente con ID " + idField.getText() + " eliminado correctamente",

"Cliente Eliminado", JOptionPane.INFORMATION\_MESSAGE);

limpiarCampos();

}

}

}

package Vista;

import Modelo.Cliente;

import Modelo.ClienteDAO;

import javax.swing.\*;

import javax.swing.border.Border;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.JTableHeader;

import javax.swing.JTextField;

import com.toedter.calendar.JDateChooser;

import Controlador.ClientesContro;

import Controlador.ReportesControlador;

import Modelo.Usuario;

import Modelo.UsuarioDAO;

import java.awt.\*;

import java.awt.event.\*;

import java.net.URL;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Objects;

import java.text.DecimalFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

public class clientes extends JFrame {

private Usuario usuario;

private String rolUsuario;

private JTable tablaClientes;

private DefaultTableModel modeloTabla;

public JButton btnAgregar, btnEditar, btnEliminar;

private Cliente nuevoCliente;

private ClientesContro controlador;

private final ClienteDAO clienteDAO;

private JTextField txtTelefono;

private JTextField txtNombre;

private JTextField txtFecha;

private JTextField txtPuntos;

public clientes(Usuario usuario, ClienteDAO clienteDAO) {

this.usuario = usuario;

this.clienteDAO = clienteDAO;

this.modeloTabla = new DefaultTableModel(new Object[] { "ID", "TELÉFONO", "NOMBRE", "ÚLTIMA COMPRA", "PUNTOS" },

0);

// Inicializar el controlador PASANDO el clienteDAO correctamente

this.controlador = new ClientesContro(this.clienteDAO, this, new Cliente(), this.modeloTabla);

UsuarioDAO usuarioDAO = new UsuarioDAO();

this.rolUsuario = usuario.getRol();

initUI();

}

private void initUI() {

setTitle("El Habanerito - Clientes");

setSize(1517, 903);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

setResizable(true);

// Inicializar el modelo de tabla

modeloTabla = new DefaultTableModel(

new Object[]{"ID", "TELÉFONO", "NOMBRE", "ÚLTIMA COMPRA", "PUNTOS"}, 0);

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.setBorder(null);

// 1. Crear componentes superiores

JPanel northContainer = new JPanel();

northContainer.setLayout(new BoxLayout(northContainer, BoxLayout.Y\_AXIS));

northContainer.add(crearPanelSuperior());

northContainer.add(crearMenuHorizontal());

northContainer.add(Box.createRigidArea(new Dimension(0, 10)));

JPanel titleWrapper = new JPanel(new FlowLayout(FlowLayout.CENTER));

titleWrapper.setOpaque(false);

titleWrapper.add(crearPanelTituloRedondeado());

northContainer.add(titleWrapper);

northContainer.add(Box.createRigidArea(new Dimension(0, 15)));

mainPanel.add(northContainer, BorderLayout.NORTH);

// 2. Crear tabla y área central

JScrollPane scrollPane = crearTablaClientes();

JPanel mainContent = new JPanel(new BorderLayout());

mainContent.setOpaque(false);

mainContent.add(scrollPane, BorderLayout.CENTER);

mainPanel.add(mainContent, BorderLayout.CENTER);

// 3. Crear panel inferior

mainPanel.add(crearPanelInferior(), BorderLayout.SOUTH);

// 4. Configurar acciones y añadir al frame

configurarAccionesBotones();

getContentPane().add(mainPanel, BorderLayout.CENTER);

// 5. Cargar datos iniciales

actualizarTablaClientes();

}

private JPanel crearPanelInferior() {

JPanel panelInferior = new JPanel(new BorderLayout());

panelInferior.setBackground(Color.PINK);

panelInferior.setBorder(null);

panelInferior.setPreferredSize(new Dimension(0, 90));

JPanel botones = crearPanelBotonesAccion();

botones.setOpaque(false); // para que el fondo rosa se vea

panelInferior.add(botones, BorderLayout.CENTER);

return panelInferior;

}

private JPanel crearPanelFondoRedondeado(Color colorFondo) {

JPanel panel = new JPanel() {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(colorFondo);

g2.fillRoundRect(0, 0, getWidth(), getHeight(), 20, 20);

}

};

panel.setOpaque(false);

panel.setLayout(new BorderLayout());

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

return panel;

}

private JPanel crearPanelSuperior() {

JPanel topPanel = new JPanel(new BorderLayout());

topPanel.setBackground(new Color(255, 198, 144));

topPanel.setPreferredSize(new Dimension(0, 60));

topPanel.setBorder(BorderFactory.createEmptyBorder(5, 15, 5, 15));

JPanel rightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 0));

rightPanel.setOpaque(false);

JButton usuarioBtn = new JButton(usuario.getUsername());

usuarioBtn.setFont(new Font("Arial", Font.BOLD, 14));

usuarioBtn.setForeground(Color.BLACK);

usuarioBtn.setContentAreaFilled(false);

usuarioBtn.setBorderPainted(false);

usuarioBtn.setFocusPainted(false);

usuarioBtn.setCursor(new Cursor(Cursor.HAND\_CURSOR));

usuarioBtn.addActionListener(e -> cambiarUsuario());

rightPanel.add(usuarioBtn);

try {

ImageIcon originalIcon = new ImageIcon("imagen\\logo.png");

Image originalImage = originalIcon.getImage();

int logoHeight = 60;

int logoWidth = (int) ((double) originalIcon.getIconWidth() / originalIcon.getIconHeight() \* logoHeight);

Image resizedImage = originalImage.getScaledInstance(logoWidth, logoHeight, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage));

rightPanel.add(logo, 0);

} catch (Exception e) {

System.err.println("Error cargando el logo: " + e.getMessage());

}

topPanel.add(rightPanel, BorderLayout.EAST);

return topPanel;

}

private JPanel crearMenuHorizontal() {

JPanel menuPanel = new JPanel(new GridLayout(1, 5));

menuPanel.setBackground(new Color(230, 230, 230));

menuPanel.setPreferredSize(new Dimension(0, 50));

menuPanel.setBorder(BorderFactory.createMatteBorder(1, 0, 1, 0, Color.GRAY));

String[] opciones = {"Productos", "Reportes", "Inventario", "Cliente","Proveedores","Usuarios", "Salir"};

for (String opcion : opciones) {

JButton btn = crearBotonMenu(opcion);

btn.addActionListener(e -> manejarAccionMenu(opcion));

menuPanel.add(btn);

}

return menuPanel;

}

private JButton crearBotonMenu(String texto) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

boton.setPreferredSize(new Dimension(0, 50));

boton.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 50));

boton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseEntered(java.awt.event.MouseEvent evt) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

public void mouseExited(java.awt.event.MouseEvent evt) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

});

return boton;

}

private void manejarAccionMenu(String opcion) {

switch (opcion) {

case "Salir":

this.dispose();

new menuprincipal(usuario).setVisible(true);

break;

case "Productos":

this.dispose();

new producto(usuario).setVisible(true);

break;

case "Reportes":

this.dispose();

reportes vistaReportes = new reportes(usuario, new ReportesControlador(null, usuario));

vistaReportes.setVisible(true); // Muestra la ventana

break;

case "Inventario":

this.dispose();

new inventario(usuario).setVisible(true);

break;

case "Cliente":

JOptionPane.showMessageDialog(this, "Ya estás en la ventana de Clientes.");

break;

case "Proveedores":

this.dispose();

new proveedores(usuario).setVisible(true);

break;

case "Usuario":

this.dispose();

new gestionUsuario(usuario).setVisible(true);

break;

}

}

private void cambiarUsuario() {

JDialog changeUserDialog = new JDialog(this, "Cambiar Usuario", true);

changeUserDialog.setSize(300, 150);

changeUserDialog.setLocationRelativeTo(this);

JPanel dialogPanel = new JPanel(new BorderLayout(10, 10));

dialogPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

JLabel instructionLabel = new JLabel("¿Desea cambiar de usuario?", SwingConstants.CENTER);

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 0));

JButton cambiarBtn = new JButton("Cambiar de Usuario");

JButton cancelarBtn = new JButton("Cancelar");

cambiarBtn.addActionListener(e -> {

changeUserDialog.dispose();

this.dispose();

new Login().setVisible(true);

});

cancelarBtn.addActionListener(e -> changeUserDialog.dispose());

buttonPanel.add(cambiarBtn);

buttonPanel.add(cancelarBtn);

dialogPanel.add(instructionLabel, BorderLayout.CENTER);

dialogPanel.add(buttonPanel, BorderLayout.SOUTH);

changeUserDialog.getContentPane().add(dialogPanel);

changeUserDialog.setVisible(true);

}

private JPanel crearPanelTituloRedondeado() {

JPanel panel = new JPanel(new BorderLayout()) {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Fondo verde con degradado

GradientPaint gradient = new GradientPaint(

0, 0, new Color(208, 244, 167), // Verde claro arriba

0, getHeight(), new Color(208, 244, 167) // Verde más oscuro abajo

);

g2d.setPaint(gradient);

g2d.fillRoundRect(0, 0, getWidth(), getHeight(), 50, 50);

// Borde sutil

g2d.setColor(new Color(0, 100, 0, 150));

g2d.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 50, 50);

}

};

panel.setOpaque(false);

panel.setPreferredSize(new Dimension(400, 60)); // Más ancho para mejor proporción

panel.setMaximumSize(new Dimension(400, 60)); // Evita que se estire demasiado

JLabel lblTitulo = new JLabel("CLIENTES");

lblTitulo.setFont(new Font("Arial", Font.BOLD, 28));

lblTitulo.setForeground(Color.BLACK);

lblTitulo.setHorizontalAlignment(SwingConstants.CENTER);

panel.add(lblTitulo, BorderLayout.CENTER);

panel.setBorder(BorderFactory.createEmptyBorder(5, 20, 5, 20));

return panel;

}

private JPanel crearCeldaRedondeada(String texto, boolean esEncabezado) {

JPanel panel = new JPanel(new BorderLayout()) {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Fondo gris para encabezados

if (esEncabezado) {

g2d.setColor(new Color(220, 220, 220)); // Gris claro

g2d.fillRoundRect(0, 0, getWidth(), getHeight(), 20, 20);

}

// Borde negro para todos

g2d.setColor(Color.BLACK);

g2d.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 20, 20);

}

};

panel.setOpaque(false); // Fondo transparente (para celdas normales)

panel.setBorder(BorderFactory.createEmptyBorder(10, 15, 10, 15));

JLabel label = new JLabel(texto, SwingConstants.CENTER);

label.setFont(new Font("Arial", esEncabezado ? Font.BOLD : Font.PLAIN,

esEncabezado ? 16 : 14));

panel.add(label, BorderLayout.CENTER);

return panel;

}

private JPanel crearFilaCliente(String telefono, String nombre,

String ultimaCompra, String puntos) {

JPanel fila = new JPanel(new GridLayout(1, 4, 10, 0));

fila.setOpaque(false);

fila.setBorder(BorderFactory.createEmptyBorder(5, 0, 5, 0));

fila.add(crearCeldaRedondeada(telefono, false));

fila.add(crearCeldaRedondeada(nombre, false));

fila.add(crearCeldaRedondeada(ultimaCompra, false));

fila.add(crearCeldaRedondeada(puntos, false));

return fila;

}

private JPanel crearEncabezadoTabla() {

JPanel encabezado = new JPanel(new GridLayout(1, 4, 10, 0));

encabezado.setOpaque(false); // Panel contenedor transparente

String[] titulos = {"TELÉFONO", "NOMBRE", "ÚLTIMA COMPRA", "PUNTOS"};

for (String titulo : titulos) {

// true indica que es encabezado (fondo gris)

encabezado.add(crearCeldaRedondeada(titulo, true));

}

return encabezado;

}

private JScrollPane crearTablaClientes() {

// 1. Crear el modelo de tabla primero si no existe

if (modeloTabla == null) {

modeloTabla = new DefaultTableModel(

new Object[]{"ID", "TELÉFONO", "NOMBRE", "FECHA", "PUNTOS"}, 0);

}

// 2. Crear tabla con el modelo

tablaClientes = new JTable(modeloTabla) {

@Override

public boolean isCellEditable(int row, int column) {

return false;

}

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setColor(new Color(0, 0, 0, 0));

g2.fillRect(0, 0, getWidth(), getHeight());

g2.dispose();

super.paintComponent(g);

}

};

// Configuración visual de la tabla

tablaClientes.setOpaque(false);

tablaClientes.setRowHeight(40);

tablaClientes.setFont(new Font("Arial", Font.PLAIN, 14));

tablaClientes.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

tablaClientes.setShowGrid(false);

tablaClientes.setIntercellSpacing(new Dimension(0, 0));

// Encabezado transparente con estilo

JTableHeader header = tablaClientes.getTableHeader();

header.setOpaque(false);

header.setBackground(new Color(0, 0, 0, 0)); // Transparente

header.setFont(new Font("Arial", Font.BOLD, 16));

header.setForeground(Color.BLACK);

// Configurar el renderizador

tablaClientes.setDefaultRenderer(Object.class, new DefaultTableCellRenderer() {

@Override

public Component getTableCellRendererComponent(JTable table, Object value,

boolean isSelected, boolean hasFocus, int row, int column) {

JPanel panel = new JPanel(new BorderLayout()) {

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Fondo según selección (semi-transparente)

if (isSelected) {

g2.setColor(new Color(208, 244, 167, 200)); // Verde claro con transparencia

} else {

g2.setColor(row % 2 == 0 ?

new Color(255, 255, 255, 50) : // Blanco muy transparente

new Color(240, 240, 240, 80)); // Gris claro transparente

}

g2.fillRoundRect(0, 0, getWidth(), getHeight(), 15, 15);

// Borde sutil semi-transparente

g2.setColor(new Color(200, 200, 200, 150));

g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 15, 15);

}

};

panel.setOpaque(false);

JLabel label = new JLabel(value == null ? "" : value.toString(), SwingConstants.CENTER);

label.setFont(table.getFont());

label.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));

label.setOpaque(false);

panel.add(label, BorderLayout.CENTER);

return panel;

}

});

// Configurar el scroll pane

JScrollPane scrollPane = new JScrollPane(tablaClientes) {

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setColor(new Color(0, 0, 0, 0));

g2.fillRect(0, 0, getWidth(), getHeight());

g2.dispose();

super.paintComponent(g);

}

};

scrollPane.setOpaque(false);

scrollPane.getViewport().setOpaque(false);

scrollPane.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

scrollPane.setViewportBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

return scrollPane;

}

private JPanel crearPanelContenido() {

JPanel contentPanel = new JPanel(new BorderLayout());

contentPanel.setOpaque(false); // Panel principal transparente

contentPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

// Panel decorativo transparente con sombra

JPanel backgroundPanel = new JPanel(new BorderLayout()) {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

// Sombra sutil (semi-transparente)

g2.setColor(new Color(0, 0, 0, 20));

g2.fillRoundRect(2, 2, getWidth()-4, getHeight()-4, 25, 25);

}

};

backgroundPanel.setOpaque(false);

backgroundPanel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

backgroundPanel.add(crearTablaClientes(), BorderLayout.CENTER);

contentPanel.add(backgroundPanel, BorderLayout.CENTER);

return contentPanel;

}

private JPanel crearPanelBotonesAccion() {

JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 10));

panel.setOpaque(false);

btnAgregar = crearBotonRedondeado("Agregar Cliente");

btnEditar = crearBotonRedondeado("Editar");

btnEliminar = crearBotonRedondeado("Eliminar");

panel.add(btnAgregar);

panel.add(btnEditar);

panel.add(btnEliminar);

return panel;

}

private JButton crearBotonRedondeado(String texto) {

JButton btn = new JButton(texto) {

@Override

protected void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(getBackground());

g2.fillRoundRect(0, 0, getWidth(), getHeight(), 30, 30);

super.paintComponent(g2);

g2.dispose();

}

@Override

protected void paintBorder(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(getBackground());

g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 30, 30);

g2.dispose();

}

};

btn.setFont(new Font("Arial", Font.BOLD, 20));

btn.setBackground(Color.LIGHT\_GRAY);

btn.setForeground(Color.BLACK);

btn.setFocusPainted(false);

btn.setContentAreaFilled(false);

btn.setOpaque(false);

btn.setBorder(BorderFactory.createEmptyBorder(10, 100, 10, 100));

return btn;

}

private void configurarAccionesBotones() {

// Asegúrate de que el botón tenga UN solo listener

for (ActionListener al : btnAgregar.getActionListeners()) {

btnAgregar.removeActionListener(al); // ✅ Elimina todos los listeners de forma segura

}

btnAgregar.addActionListener(e ->

controlador.agregarCliente());

btnEditar.addActionListener(e -> {

controlador.editarCliente(); // Llamada directa sin validaciones

});

btnEliminar.addActionListener(e -> {

if (btnEliminar.isEnabled()) { // <- Previene doble ejecución

controlador.eliminarCliente();

}

});

}

private void agregarCliente() {

if (controlador == null) {

mostrarError("Controlador no inicializado");

return;

}

Cliente nuevoCliente = new Cliente();

if (mostrarFormularioCliente(nuevoCliente)) {

try {

controlador.agregarCliente();

actualizarTablaClientes();

mostrarConfirmacion("Cliente agregado exitosamente");

} catch (Exception ex) {

mostrarError("Error al agregar cliente: " + ex.getMessage());

}

}

}

private void editarCliente() {

if (controlador == null) {

mostrarError("Controlador no inicializado");

return;

}

int filaSeleccionada = tablaClientes.getSelectedRow();

if (filaSeleccionada < 0) {

mostrarAdvertencia("Seleccione un cliente para editar");

return;

}

String telefono = (String) tablaClientes.getValueAt(filaSeleccionada, 0);

Cliente cliente = controlador.buscarClientePorTelefono(telefono);

if (cliente == null) {

mostrarError("Cliente no encontrado");

return;

}

// Guardar copia para verificar cambios

Cliente copiaOriginal = new Cliente();

if (mostrarFormularioCliente(cliente)) {

if (!cliente.equals(copiaOriginal)) {

try {

controlador.actualizarCliente(cliente);

actualizarTablaClientes();

mostrarConfirmacion("Cliente actualizado exitosamente");

} catch (Exception ex) {

mostrarError("Error al actualizar cliente: " + ex.getMessage());

// Revertir cambios en caso de error

cliente.copiarDe(copiaOriginal);

}

} else {

mostrarInformacion("No se realizaron cambios");

}

}

}

private void eliminarCliente() {

if (controlador == null) {

mostrarError("Controlador no inicializado");

return;

}

int filaSeleccionada = tablaClientes.getSelectedRow();

if (filaSeleccionada < 0) {

mostrarAdvertencia("Seleccione un cliente para eliminar");

return;

}

String telefono = (String) tablaClientes.getValueAt(filaSeleccionada, 0);

if (mostrarConfirmacion("¿Está seguro de eliminar este cliente?")) {

try {

controlador.eliminarCliente();

actualizarTablaClientes();

mostrarConfirmacion("Cliente eliminado exitosamente");

} catch (Exception ex) {

mostrarError("Error al eliminar cliente: " + ex.getMessage());

}

}

}

public void actualizarTablaClientes() {

modeloTabla.setRowCount(0); // Limpiar tabla

try {

// Obtener clientes desde el controlador

List<Cliente> clientes = controlador.obtenerTodosClientes();

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

for (Cliente cliente : clientes) {

// Formatear la fecha para mostrarla correctamente

String fechaFormateada = "";

if (cliente.getUltimaCompra() != null && !cliente.getUltimaCompra().isEmpty()) {

try {

Date fecha = sdf.parse(cliente.getUltimaCompra());

fechaFormateada = sdf.format(fecha);

} catch (ParseException e) {

fechaFormateada = cliente.getUltimaCompra();

}

}

modeloTabla.addRow(new Object[]{

cliente.getId(),

cliente.getTelefono(),

cliente.getNombre(),

fechaFormateada,

cliente.getPuntos()

});

}

} catch (Exception e) {

mostrarError("Error al cargar clientes: " + e.getMessage());

e.printStackTrace();

}

}

public JButton getBtnAgregar() {

return btnAgregar;

}

public JButton getBtnEditar() {

return btnEditar;

}

public JButton getBtnEliminar() {

return btnEliminar;

}

public boolean mostrarFormularioCliente(Cliente cliente) {

JDialog dialog = new JDialog(this, "Nuevo Cliente", true);

dialog.setLayout(new BorderLayout());

dialog.setPreferredSize(new Dimension(500, 400));

dialog.setResizable(false);

// Panel principal

JPanel mainPanel = new JPanel(new BorderLayout(10, 10));

mainPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// Título

JLabel titleLabel = new JLabel("NUEVO CLIENTE", SwingConstants.CENTER);

titleLabel.setFont(new Font("Arial", Font.BOLD, 18));

mainPanel.add(titleLabel, BorderLayout.NORTH);

// Panel de formulario

JPanel formPanel = new JPanel(new GridBagLayout());

formPanel.setBorder(BorderFactory.createTitledBorder("Información del Cliente"));

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(5, 5, 5, 5);

gbc.anchor = GridBagConstraints.WEST;

gbc.fill = GridBagConstraints.HORIZONTAL;

// Componentes del formulario

JLabel lblTelefono = new JLabel("Teléfono:");

txtTelefono = new JTextField(20);

txtTelefono.setText(cliente.getTelefono() != null ? cliente.getTelefono() : "");

JLabel lblNombre = new JLabel("Nombre:");

txtNombre = new JTextField(20);

txtNombre.setText(cliente.getNombre() != null ? cliente.getNombre() : "");

JLabel lblFecha = new JLabel("Última Compra:");

JDateChooser dateChooser = new JDateChooser();

dateChooser.setDateFormatString("dd/MM/yyyy");

dateChooser.setPreferredSize(new Dimension(150, 25));

// Configurar fecha inicial si existe

if (cliente.getUltimaCompra() != null && !cliente.getUltimaCompra().isEmpty()) {

try {

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

Date fecha = sdf.parse(cliente.getUltimaCompra());

dateChooser.setDate(fecha);

} catch (ParseException e) {

dateChooser.setDate(new Date()); // Fecha actual por defecto

}

} else {

dateChooser.setDate(new Date()); // Fecha actual por defecto

}

JLabel lblPuntos = new JLabel("Puntos:");

txtPuntos = new JTextField(20);

txtPuntos.setText(String.valueOf(cliente.getPuntos()));

txtPuntos.setEditable(false);

// Agregar componentes al formulario

gbc.gridx = 0; gbc.gridy = 0;

formPanel.add(lblTelefono, gbc);

gbc.gridx = 1;

formPanel.add(txtTelefono, gbc);

gbc.gridx = 0; gbc.gridy = 1;

formPanel.add(lblNombre, gbc);

gbc.gridx = 1;

formPanel.add(txtNombre, gbc);

gbc.gridx = 0; gbc.gridy = 2;

formPanel.add(lblFecha, gbc);

gbc.gridx = 1;

formPanel.add(dateChooser, gbc);

gbc.gridx = 0; gbc.gridy = 3;

formPanel.add(lblPuntos, gbc);

gbc.gridx = 1;

formPanel.add(txtPuntos, gbc);

mainPanel.add(formPanel, BorderLayout.CENTER);

// Panel de botones

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JButton btnCancelar = new JButton("Cancelar");

JButton btnGuardar = new JButton("Guardar");

// Estilo de botones

btnGuardar.setBackground(new Color(76, 175, 80)); // Verde

btnGuardar.setForeground(Color.WHITE);

btnCancelar.setBackground(new Color(244, 67, 54)); // Rojo

btnCancelar.setForeground(Color.WHITE);

final boolean[] resultado = {false};

// Acción para el botón Guardar

btnGuardar.addActionListener(e -> {

if (!validarDatos()) return;

// Obtener fecha del JDateChooser y formatear para Access

SimpleDateFormat sdfAccess = new SimpleDateFormat("yyyy-MM-dd");

String fechaParaAccess = sdfAccess.format(dateChooser.getDate());

// Asignar valores al objeto cliente

cliente.setTelefono(txtTelefono.getText().trim());

cliente.setNombre(txtNombre.getText().trim());

cliente.setUltimaCompra(fechaParaAccess);

try {

cliente.setPuntos(Integer.parseInt(txtPuntos.getText()));

} catch (NumberFormatException ex) {

cliente.setPuntos(0);

}

resultado[0] = true;

dialog.dispose();

});

// Acción para el botón Cancelar

btnCancelar.addActionListener(e -> {

resultado[0] = false;

dialog.dispose();

});

buttonPanel.add(btnCancelar);

buttonPanel.add(btnGuardar);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

// Configurar comportamiento de la tecla Enter

dialog.getRootPane().setDefaultButton(btnGuardar);

// Mostrar diálogo

dialog.add(mainPanel);

dialog.pack();

dialog.setLocationRelativeTo(this);

// Enfocar el campo de teléfono al abrir

SwingUtilities.invokeLater(() -> txtTelefono.requestFocusInWindow());

dialog.setVisible(true);

return resultado[0];

}

private boolean validarDatos() {

if (txtTelefono.getText().trim().isEmpty()) {

mostrarError("El teléfono es requerido");

return false;

}

if (txtNombre.getText().trim().isEmpty()) {

mostrarError("El nombre es requerido");

return false;

}

if (!txtTelefono.getText().matches("\\d{7,15}")) {

mostrarError("Teléfono debe contener solo números (7-15 dígitos)");

return false;

}

return true;

}

public boolean mostrarConfirmacion(String mensaje) {

int opcion = JOptionPane.showConfirmDialog(

this,

mensaje,

"Confirmar eliminación",

JOptionPane.YES\_NO\_OPTION,

JOptionPane.WARNING\_MESSAGE

);

return opcion == JOptionPane.YES\_OPTION;

}

public void mostrarError(String mensaje) {

// Cargar icono desde recursos

ImageIcon errorIcon = cargarIcono("/images/error\_icon.png");

JOptionPane.showMessageDialog(

this,

mensaje,

"Error",

JOptionPane.ERROR\_MESSAGE,

errorIcon != null ? errorIcon : UIManager.getIcon("OptionPane.errorIcon"));

}

private ImageIcon cargarIcono(String ruta) {

try {

URL imgURL = getClass().getResource(ruta);

if (imgURL != null) {

return new ImageIcon(imgURL);

}

} catch (Exception e) {

System.err.println("Error al cargar icono: " + e.getMessage());

}

return null;

}

public void mostrarAdvertencia(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Advertencia", JOptionPane.WARNING\_MESSAGE);

}

public void mostrarInformacion(String mensaje) {

JOptionPane.showMessageDialog(this, mensaje, "Información", JOptionPane.INFORMATION\_MESSAGE);

}

public JTable getTablaClientes() {

if (tablaClientes == null) {

throw new IllegalStateException("Tabla no inicializada");

}

return tablaClientes;

}

public boolean mostrarFormularioEdicion(Cliente cliente) {

if (cliente == null) {

throw new IllegalArgumentException("Cliente no puede ser nulo");

}

if (cliente.getTelefono() == null) {

cliente.setTelefono(""); // Valor por defecto

}

// Configuración del diálogo

JDialog dialog = new JDialog(this, "Editar Cliente", true);

dialog.setDefaultCloseOperation(JDialog.DISPOSE\_ON\_CLOSE);

dialog.setLayout(new BorderLayout());

dialog.setPreferredSize(new Dimension(500, 350));

dialog.setResizable(false);

// Panel principal

JPanel mainPanel = new JPanel(new BorderLayout(10, 10));

mainPanel.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));

// Panel de título

JLabel titleLabel = new JLabel("EDITAR CLIENTE", SwingConstants.CENTER);

titleLabel.setFont(new Font("Arial", Font.BOLD, 18));

mainPanel.add(titleLabel, BorderLayout.NORTH);

// Panel de formulario

JPanel formPanel = new JPanel(new GridBagLayout());

formPanel.setBorder(BorderFactory.createTitledBorder("Información del Cliente"));

GridBagConstraints gbc = new GridBagConstraints();

gbc.insets = new Insets(5, 5, 5, 5);

gbc.anchor = GridBagConstraints.WEST;

gbc.fill = GridBagConstraints.HORIZONTAL;

// Campos del formulario

JLabel lblId = new JLabel("ID:");

JTextField txtId = new JTextField(20);

txtId.setText(cliente.getId() != null ? cliente.getId() : "");

txtId.setEditable(false);

JLabel lblTelefono = new JLabel("Teléfono:");

JTextField txtTelefono = new JTextField(20);

txtTelefono.setText(cliente.getTelefono() != null ? cliente.getTelefono() : "");

JLabel lblNombre = new JLabel("Nombre:");

JTextField txtNombre = new JTextField(20);

txtNombre.setText(cliente.getNombre() != null ? cliente.getNombre() : "");

JLabel lblFecha = new JLabel("Última Compra:");

JTextField txtFecha = new JTextField(20);

txtFecha.setText(cliente.getUltimaCompra() != null ? cliente.getUltimaCompra() : "");

JLabel lblPuntos = new JLabel("Puntos:");

JTextField txtPuntos = new JTextField(20);

txtPuntos.setText(String.valueOf(cliente.getPuntos()));

txtPuntos.setEditable(false);

// Añadir componentes al formulario

gbc.gridx = 0; gbc.gridy = 0;

formPanel.add(lblId, gbc);

gbc.gridx = 1;

formPanel.add(txtId, gbc);

gbc.gridx = 0; gbc.gridy = 1;

formPanel.add(lblTelefono, gbc);

gbc.gridx = 1;

formPanel.add(txtTelefono, gbc);

gbc.gridx = 0; gbc.gridy = 2;

formPanel.add(lblNombre, gbc);

gbc.gridx = 1;

formPanel.add(txtNombre, gbc);

gbc.gridx = 0; gbc.gridy = 3;

formPanel.add(lblFecha, gbc);

gbc.gridx = 1;

formPanel.add(txtFecha, gbc);

gbc.gridx = 0; gbc.gridy = 4;

formPanel.add(lblPuntos, gbc);

gbc.gridx = 1;

formPanel.add(txtPuntos, gbc);

mainPanel.add(formPanel, BorderLayout.CENTER);

// Panel de botones

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JButton btnCancelar = new JButton("Cancelar");

JButton btnGuardar = new JButton("Guardar Cambios");

// Estilo de botones

btnGuardar.setBackground(new Color(76, 175, 80)); // Verde

btnGuardar.setForeground(Color.WHITE);

btnCancelar.setBackground(new Color(244, 67, 54)); // Rojo

btnCancelar.setForeground(Color.WHITE);

final boolean[] resultado = {false};

btnGuardar.addActionListener(e -> {

// Validar campos

if (txtTelefono.getText().trim().isEmpty()) {

mostrarError("El teléfono es requerido");

return;

}

// Verificar si hubo cambios reales

boolean cambios = !txtTelefono.getText().trim().equals(cliente.getTelefono()) ||

!txtNombre.getText().trim().equals(cliente.getNombre()) ||

!txtFecha.getText().trim().equals(cliente.getUltimaCompra());

if (!cambios) {

mostrarInformacion("No se realizaron cambios");

dialog.dispose(); // Cerrar el diálogo

return;

}

// Actualizar objeto cliente

cliente.setTelefono(txtTelefono.getText().trim());

cliente.setNombre(txtNombre.getText().trim());

cliente.setUltimaCompra(txtFecha.getText().trim());

resultado[0] = true;

dialog.dispose(); // Cerrar definitivamente

});

btnCancelar.addActionListener(e -> {

dialog.dispose();

});

buttonPanel.add(btnCancelar);

buttonPanel.add(btnGuardar);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

// Configurar comportamiento de la tecla Enter

dialog.getRootPane().setDefaultButton(btnGuardar);

// Mostrar diálogo

dialog.add(mainPanel);

dialog.pack();

dialog.setLocationRelativeTo(this);

// Enfocar el campo de teléfono al abrir

SwingUtilities.invokeLater(() -> txtTelefono.requestFocusInWindow());

dialog.setVisible(true);

return resultado[0];

}

public int getTablaSeleccionada() {

return tablaClientes.getSelectedRow();

}

public String getSelectedClienteId() {

try {

int fila = tablaClientes.getSelectedRow();

if (fila >= 0) {

return tablaClientes.getValueAt(fila, 0).toString();// Columna 0 es ID

}

} catch (Exception e) {

// Silenciar cualquier error

}

return null;

}

public int getSelectedRow() {

return tablaClientes.getSelectedRow();

}

// Método auxiliar para obtener modelo

public DefaultTableModel getModelo() {

return (DefaultTableModel) tablaClientes.getModel();

}

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.}

# Proveedores

package Modelo;

import java.sql.Date;

import java.sql.Timestamp;

public class Proveedor {

private String id;

private String nombre;

private String telefono;

private String direccion;

private String productoSuministrado;

private Timestamp ultimaVisita;

public Proveedor(String id, String nombre, String telefono, String direccion,

String productoSuministrado, Timestamp ultimaVisita) {

this.id = id;

this.nombre = nombre;

this.telefono = telefono;

this.direccion = direccion;

this.productoSuministrado = productoSuministrado;

this.ultimaVisita = ultimaVisita;

}

// Getters y Setters

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public String getTelefono() {

return telefono;

}

public void setTelefono(String telefono) {

this.telefono = telefono;

}

public String getDireccion() {

return direccion;

}

public void setDireccion(String direccion) {

this.direccion = direccion;

}

public String getProductoSuministrado() {

return productoSuministrado;

}

public void setProductoSuministrado(String productoSuministrado) {

this.productoSuministrado = productoSuministrado;

}

public Timestamp getUltimaVisita() {

return ultimaVisita;

}

public void setUltimaVisita(Timestamp ultimaVisita) {

this.ultimaVisita = ultimaVisita;

}

@Override

public String toString() {

return "Proveedor{" +

"id='" + id + '\'' +

", nombre='" + nombre + '\'' +

", telefono='" + telefono + '\'' +

", direccion='" + direccion + '\'' +

", productoSuministrado='" + productoSuministrado + ", ultimaVisita='" + ultimaVisita + '\'' +

'}';

}

}

package Controlador;

import Modelo.Proveedor;

import Modelo.ProveedorDAO;

import Vista.proveedores;

import java.sql.Timestamp;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import javax.swing.table.DefaultTableModel;

public class ProveedoresContro {

private final ProveedorDAO proveedorDAO;

private final proveedores vista;

private final DefaultTableModel modeloTabla;

public ProveedoresContro(ProveedorDAO proveedorDAO, proveedores vista, DefaultTableModel modeloTabla) {

this.proveedorDAO = proveedorDAO;

this.vista = vista;

this.modeloTabla = modeloTabla;

}

public boolean agregarProveedor(Proveedor nuevoProveedor) {

if (proveedorDAO.agregarProveedor(nuevoProveedor)) {

actualizarTablaProveedores();

vista.mostrarMensaje("Proveedor registrado con ID: " + nuevoProveedor.getId(), 1);

return true;

}

vista.mostrarMensaje("Error al registrar proveedor", 3);

return false;

}

public boolean editarProveedor(Proveedor proveedor) {

if (proveedorDAO.actualizarProveedor(proveedor)) {

actualizarTablaProveedores();

vista.mostrarMensaje("Proveedor actualizado exitosamente", 1);

return true;

}

vista.mostrarMensaje("Error al actualizar proveedor", 3);

return false;

}

public boolean eliminarProveedor(String id) {

if (proveedorDAO.eliminarProveedor(id)) {

actualizarTablaProveedores();

vista.mostrarMensaje("Proveedor eliminado exitosamente", 1);

return true;

}

vista.mostrarMensaje("Error al eliminar proveedor", 3);

return false;

}

public Proveedor buscarProveedorPorId(String id) {

return proveedorDAO.buscarProveedorPorId(id);

}

public void actualizarTablaProveedores() {

try {

DefaultTableModel model = (DefaultTableModel) vista.getTablaProveedores().getModel();

model.setRowCount(0); // Limpiar tabla

List<Proveedor> proveedores = proveedorDAO.obtenerTodosProveedores();

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

for (Proveedor p : proveedores) {

model.addRow(new Object[]{

p.getId(),

p.getNombre(),

p.getTelefono(),

p.getDireccion(),

p.getProductoSuministrado(),

p.getUltimaVisita() != null ? sdf.format(p.getUltimaVisita()) : ""

});

}

// Notificar cambios y actualizar visualización

model.fireTableDataChanged();

vista.getTablaProveedores().repaint();

} catch (Exception e) {

vista.mostrarMensaje("Error al actualizar tabla: " + e.getMessage(), 3);

e.printStackTrace();

}

}

public void registrarVisitaProveedor(String idProveedor) {

Proveedor proveedor = proveedorDAO.buscarProveedorPorId(idProveedor);

if (proveedor != null) {

proveedor.setUltimaVisita(new Timestamp(System.currentTimeMillis()));

if (proveedorDAO.actualizarProveedor(proveedor)) {

actualizarTablaProveedores();

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

vista.mostrarMensaje("Visita registrada para " + proveedor.getNombre() + " - " +

sdf.format(proveedor.getUltimaVisita()), 1);

} else {

vista.mostrarMensaje("Error al registrar visita", 3);

}

}

}

}

package Modelo;

import ConexionBD.ConexionAccess;

import Vista.proveedores;

import java.sql.\*;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.List;

public class ProveedorDAO {

public ProveedorDAO() {

}

public boolean agregarProveedor(Proveedor proveedor) {

String sql = "INSERT INTO Proveedores (ID, Nombre, Telefono, direccion, producto\_suministrado, ultima\_visita) VALUES (?, ?, ?, ?, ?, ?)";

// Obtener nueva conexión para esta operación

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

String idProveedor = proveedor.getId();

if (idProveedor == null || idProveedor.isEmpty()) {

idProveedor = "PRV-" + System.currentTimeMillis();

}

pstmt.setString(1, idProveedor);

pstmt.setString(2, proveedor.getNombre());

pstmt.setString(3, proveedor.getTelefono());

pstmt.setString(4, proveedor.getDireccion() != null ? proveedor.getDireccion() : "");

pstmt.setString(5, proveedor.getProductoSuministrado() != null ? proveedor.getProductoSuministrado() : "");

if (proveedor.getUltimaVisita() != null) {

pstmt.setTimestamp(6, proveedor.getUltimaVisita());

} else {

pstmt.setTimestamp(6, new Timestamp(System.currentTimeMillis()));

}

int affectedRows = pstmt.executeUpdate();

if (affectedRows > 0) {

proveedor.setId(idProveedor);

return true;

}

} catch (SQLException e) {

System.err.println("Error detallado al agregar proveedor:");

System.err.println("SQL: " + sql);

System.err.println("Mensaje: " + e.getMessage());

e.printStackTrace();

}

return false;

}

public List<Proveedor> obtenerTodosProveedores() {

List<Proveedor> proveedores = new ArrayList<>();

String query = "SELECT \* FROM proveedores";

try (Connection conn = ConexionAccess.conectar();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query)) {

while (rs.next()) {

Timestamp ultimaVisita = rs.getTimestamp("ultima\_visita");

if (ultimaVisita == null) {

ultimaVisita = new Timestamp(System.currentTimeMillis());

}

Proveedor p = new Proveedor(

rs.getString("id"),

rs.getString("nombre"),

rs.getString("telefono"),

rs.getString("direccion"),

rs.getString("producto\_suministrado"),

ultimaVisita

);

proveedores.add(p);

}

} catch (SQLException e) {

System.err.println("Error al obtener proveedores: " + e.getMessage());

e.printStackTrace();

}

return proveedores;

}

public Proveedor buscarProveedorPorId(String id) {

String query = "SELECT \* FROM proveedores WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setString(1, id);

try (ResultSet rs = pstmt.executeQuery()) {

if (rs.next()) {

return new Proveedor(

rs.getString("id"),

rs.getString("nombre"),

rs.getString("telefono"),

rs.getString("direccion"),

rs.getString("producto\_suministrado"),

rs.getTimestamp("ultima\_visita")

);

}

}

} catch (SQLException e) {

System.err.println("Error al buscar proveedor: " + e.getMessage());

}

return null;

}

public boolean actualizarProveedor(Proveedor proveedor) {

String sql = "UPDATE proveedores SET nombre = ?, telefono = ?, direccion = ?, producto\_suministrado = ?, ultima\_visita = ? WHERE id = ?";

// Obtener nueva conexión para esta operación

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, proveedor.getNombre());

pstmt.setString(2, proveedor.getTelefono());

pstmt.setString(3, proveedor.getDireccion());

pstmt.setString(4, proveedor.getProductoSuministrado());

pstmt.setTimestamp(5, proveedor.getUltimaVisita());

pstmt.setString(6, proveedor.getId());

int affectedRows = pstmt.executeUpdate();

return affectedRows > 0;

} catch (SQLException e) {

System.err.println("Error al actualizar proveedor: " + e.getMessage());

e.printStackTrace();

return false;

}

}

public boolean eliminarProveedor(String id) {

String sql = "DELETE FROM proveedores WHERE id = ?";

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, id);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

System.err.println("Error al eliminar proveedor: " + e.getMessage());

e.printStackTrace();

return false;

}

}

public List<Proveedor> obtenerProveedoresConVisitaReciente() {

Calendar cal = Calendar.getInstance();

cal.add(Calendar.MONTH, -1);

Timestamp haceUnMes = new Timestamp(cal.getTimeInMillis());

String query = "SELECT \* FROM proveedores WHERE ultima\_visita >= ?";

List<Proveedor> proveedores = new ArrayList<>();

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setTimestamp(1, haceUnMes);

try (ResultSet rs = pstmt.executeQuery()) {

while (rs.next()) {

proveedores.add(new Proveedor(

rs.getString("id"),

rs.getString("nombre"),

rs.getString("telefono"),

rs.getString("direccion"),

rs.getString("producto\_suministrado"),

rs.getTimestamp("ultima\_visita")

));

}

}

} catch (SQLException e) {

System.err.println("Error al obtener proveedores con visita reciente: " + e.getMessage());

}

return proveedores;

}

public List<Proveedor> obtenerProveedoresSinVisitaReciente() {

Calendar cal = Calendar.getInstance();

cal.add(Calendar.MONTH, -1);

Timestamp haceUnMes = new Timestamp(cal.getTimeInMillis());

String query = "SELECT \* FROM proveedores WHERE ultima\_visita IS NULL OR ultima\_visita < ?";

List<Proveedor> proveedores = new ArrayList<>();

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setTimestamp(1, haceUnMes);

try (ResultSet rs = pstmt.executeQuery()) {

while (rs.next()) {

proveedores.add(new Proveedor(

rs.getString("id"),

rs.getString("nombre"),

rs.getString("telefono"),

rs.getString("direccion"),

rs.getString("producto\_suministrado"),

rs.getTimestamp("ultima\_visita")

));

}

}

} catch (SQLException e) {

System.err.println("Error al obtener proveedores sin visita reciente: " + e.getMessage());

}

return proveedores;

}

public List<Proveedor> buscarProveedoresPorProducto(String producto) {

String query = "SELECT \* FROM proveedores WHERE producto\_suministrado LIKE ?";

List<Proveedor> proveedores = new ArrayList<>();

try (Connection conn = ConexionAccess.conectar();

PreparedStatement pstmt = conn.prepareStatement(query)) {

pstmt.setString(1, "%" + producto + "%");

try (ResultSet rs = pstmt.executeQuery()) {

while (rs.next()) {

proveedores.add(new Proveedor(

rs.getString("id"),

rs.getString("nombre"),

rs.getString("telefono"),

rs.getString("direccion"),

rs.getString("producto\_suministrado"),

rs.getTimestamp("ultima\_visita")

));

}

}

} catch (SQLException e) {

System.err.println("Error al buscar proveedores por producto: " + e.getMessage());

}

return proveedores;

}

}

package Vista;

import java.awt.\*;

import java.awt.event.\*;

import java.sql.Timestamp;

import javax.swing.\*;

import javax.swing.border.\*;

import javax.swing.table.\*;

import com.toedter.calendar.JDateChooser;

import Modelo.Usuario;

import Modelo.Proveedor;

import Modelo.ProveedorDAO;

import Controlador.ProveedoresContro;

import Controlador.ReportesControlador;

import java.text.SimpleDateFormat;

import java.util.Date;

public class proveedores extends JFrame {

private Usuario usuario;

private JTable tablaProveedores;

private DefaultTableModel modeloTabla;

private JButton btnAgregar, btnEditar, btnEliminar;

private ProveedoresContro controlador;

private final ProveedorDAO proveedorDAO;

public proveedores(Usuario usuario) {

this.usuario = usuario;

this.proveedorDAO = new ProveedorDAO();

this.modeloTabla = new DefaultTableModel(

new Object[]{"ID", "Nombre", "Teléfono", "Dirección", "Producto", "Última Visita"}, 0);

this.controlador = new ProveedoresContro(proveedorDAO, this, modeloTabla);

initUI();

cargarDatosIniciales();

}

private void cargarDatosIniciales() {

SwingUtilities.invokeLater(() -> {

try {

controlador.actualizarTablaProveedores();

} catch (Exception e) {

mostrarMensaje("Error al cargar proveedores: " + e.getMessage(), 3);

e.printStackTrace();

}

});

}

private void initUI() {

setTitle("El Habanerito - Proveedores");

setSize(1200, 800);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

// Panel principal

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.setBorder(new EmptyBorder(0, 0, 0, 0));

mainPanel.setBackground(new Color(240, 240, 240));

// 1. Panel superior (logo + usuario)

JPanel topPanel = crearPanelSuperior();

// 2. Menú horizontal

JPanel menuPanel = crearMenuHorizontal();

// 3. Panel de tabla

tablaProveedores = new JTable(modeloTabla);

configurarTabla();

JScrollPane scrollPane = new JScrollPane(tablaProveedores);

scrollPane.setBorder(BorderFactory.createCompoundBorder(

new EmptyBorder(15, 15, 15, 15),

BorderFactory.createLineBorder(new Color(200, 200, 200), 1, true)

));

// 4. Panel inferior rosa con botones

JPanel bottomPanel = crearPanelInferior();

// Panel contenedor para menu + tabla

JPanel centerPanel = new JPanel(new BorderLayout());

centerPanel.add(menuPanel, BorderLayout.NORTH);

centerPanel.add(scrollPane, BorderLayout.CENTER);

// Ensamblar componentes

mainPanel.add(topPanel, BorderLayout.NORTH);

mainPanel.add(centerPanel, BorderLayout.CENTER); // Aquí agregamos el panel combinado

mainPanel.add(bottomPanel, BorderLayout.SOUTH);

getContentPane().add(mainPanel);

}

private JPanel crearPanelSuperior() {

JPanel topPanel = new JPanel(new BorderLayout());

topPanel.setBackground(new Color(255, 198, 144));

topPanel.setPreferredSize(new Dimension(0, 60));

topPanel.setBorder(BorderFactory.createEmptyBorder(5, 15, 5, 15));

// Panel contenedor para logo + usuario (ahora en el mismo FlowLayout)

JPanel logoUserPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 5, 0));

logoUserPanel.setOpaque(false);

// Logo

try {

ImageIcon originalIcon = new ImageIcon("imagen\\logo.png");

Image resizedImage = originalIcon.getImage().getScaledInstance(60, 60, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage));

logoUserPanel.add(logo);

} catch (Exception e) {

logoUserPanel.add(new JLabel("Logo"));

}

// Usuario (pegado al logo)

JLabel lblUsuario = new JLabel(usuario.getUsername());

lblUsuario.setFont(new Font("Arial", Font.BOLD, 14));

lblUsuario.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

lblUsuario.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

mostrarMenuUsuario();

}

});

// Espaciado mínimo entre logo y usuario

logoUserPanel.add(Box.createRigidArea(new Dimension(5, 0)));

logoUserPanel.add(lblUsuario);

// Alinear todo a la derecha

JPanel containerPanel = new JPanel(new BorderLayout());

containerPanel.setOpaque(false);

containerPanel.add(logoUserPanel, BorderLayout.EAST);

topPanel.add(containerPanel, BorderLayout.CENTER);

return topPanel;

}

private JPanel crearMenuHorizontal() {

JPanel menuPanel = new JPanel(new GridLayout(1, 7));

menuPanel.setBackground(new Color(230, 230, 230));

menuPanel.setPreferredSize(new Dimension(0, 50));

menuPanel.setBorder(BorderFactory.createMatteBorder(1, 0, 1, 0, Color.GRAY));

String[] opciones = {"Productos", "Reportes", "Inventario", "Cliente", "Proveedores", "Usuarios", "Salir"};

for (String opcion : opciones) {

JButton btn = crearBotonMenu(opcion, opcion.equals("Proveedores"));

btn.addActionListener(e -> manejarAccionMenu(opcion));

menuPanel.add(btn);

}

return menuPanel;

}

private JButton crearBotonMenu(String texto, boolean esActivo) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

boton.setPreferredSize(new Dimension(0, 50));

boton.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 50));

// Configuración de colores según estado

if (esActivo) {

// Estilo para el botón activo (Proveedores)

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.BLACK);

} else {

// Estilo para botones inactivos

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

// Comportamiento al pasar el mouse (solo para botones inactivos)

if (!esActivo) {

boton.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

@Override

public void mouseExited(MouseEvent e) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

});

}

return boton;

}

private JButton crearBotonMenu(String texto) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 0, 10, 0));

boton.setPreferredSize(new Dimension(0, 50));

boton.setMaximumSize(new Dimension(Integer.MAX\_VALUE, 50));

// Guardar el color original

Color originalBg = boton.getBackground();

Color originalFg = boton.getForeground();

boton.addMouseListener(new java.awt.event.MouseAdapter() {

public void mouseEntered(java.awt.event.MouseEvent evt) {

// Solo cambiar si no es el botón activo

if (!boton.getBackground().equals(new Color(216, 237, 88))) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

}

public void mouseExited(java.awt.event.MouseEvent evt) {

// Solo cambiar si no es el botón activo

if (!boton.getBackground().equals(new Color(216, 237, 88))) {

boton.setBackground(originalBg);

boton.setForeground(originalFg);

}

}

});

return boton;

}

private void manejarAccionMenu(String opcion) {

switch (opcion) {

case "Salir":

this.dispose();

new menuprincipal(usuario).setVisible(true);

break;

case "Productos":

this.dispose();

new producto(usuario).setVisible(true);

break;

case "Reportes":

this.dispose();

// Crear una nueva instancia de ReportesControlador

ReportesControlador reportesControlador = new ReportesControlador(null, usuario);

// Crear y mostrar la ventana de reportes

reportes vistaReportes = new reportes(usuario, new ReportesControlador(null, usuario));

vistaReportes.setVisible(true); // Muestra la ventana

break;

case "Inventario":

this.dispose();

new inventario(usuario).setVisible(true);

break;

case "Cliente":

this.dispose();

new clientes(usuario, null).setVisible(true);

break;

case "Proveedores":

JOptionPane.showMessageDialog(this, "Ya estás en la ventana de Proveedores.");

break;

case "Usuarios":

this.dispose();

new gestionUsuario(usuario).setVisible(true);

break;

}

}

private JPanel crearPanelInferior() {

JPanel panel = new JPanel(new BorderLayout());

panel.setPreferredSize(new Dimension(0, 100));

panel.setBackground(new Color(255, 182, 193)); // Color rosa

panel.setBorder(BorderFactory.createMatteBorder(1, 0, 0, 0, Color.GRAY));

// Panel para centrar los botones

JPanel buttonPanel = new JPanel();

buttonPanel.setOpaque(false);

// Crear botones con bordes redondeados

btnAgregar = crearBotonAccion("Agregar");

btnEditar = crearBotonAccion("Editar");

btnEliminar = crearBotonAccion("Eliminar");

// Acciones

btnAgregar.addActionListener(e -> agregarProveedor());

btnEditar.addActionListener(e -> editarProveedor());

btnEliminar.addActionListener(e -> eliminarProveedor());

// Espaciado entre botones

buttonPanel.add(btnAgregar);

buttonPanel.add(Box.createRigidArea(new Dimension(20, 0)));

buttonPanel.add(btnEditar);

buttonPanel.add(Box.createRigidArea(new Dimension(20, 0)));

buttonPanel.add(btnEliminar);

panel.add(buttonPanel, BorderLayout.CENTER);

return panel;

}

private JButton crearBotonAccion(String texto) {

JButton boton = new JButton(texto) {

@Override

protected void paintBorder(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(Color.DARK\_GRAY);

g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 15, 15);

g2.dispose();

}

};

boton.setFont(new Font("Arial", Font.BOLD, 16));

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

boton.setFocusPainted(false);

boton.setContentAreaFilled(false);

boton.setOpaque(true);

boton.setPreferredSize(new Dimension(255, 50));

boton.setBorder(BorderFactory.createEmptyBorder(5, 20, 5, 20));

boton.addMouseListener(new MouseAdapter() {

public void mouseEntered(MouseEvent e) {

boton.setBackground(Color.LIGHT\_GRAY);

boton.setForeground(Color.WHITE);

}

public void mouseExited(MouseEvent e) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

});

return boton;

}

public void configurarTabla() {

tablaProveedores.setRowHeight(30);

tablaProveedores.setFont(new Font("Arial", Font.PLAIN, 14));

tablaProveedores.getTableHeader().setFont(new Font("Arial", Font.BOLD, 14));

tablaProveedores.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

// Renderizador para fechas

DefaultTableCellRenderer dateRenderer = new DefaultTableCellRenderer() {

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");

@Override

public Component getTableCellRendererComponent(JTable table, Object value,

boolean isSelected, boolean hasFocus, int row, int column) {

if (value instanceof Timestamp) {

value = sdf.format((Timestamp)value);

}

return super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

}

};

dateRenderer.setHorizontalAlignment(JLabel.CENTER);

// Aplicar renderizador a todas las columnas para consistencia

for (int i = 0; i < tablaProveedores.getColumnCount(); i++) {

tablaProveedores.getColumnModel().getColumn(i).setCellRenderer(dateRenderer);

}

}

private void agregarProveedor() {

// Generar un ID temporal único

String nuevoId = "PRV-" + System.currentTimeMillis();

Proveedor nuevoProveedor = new Proveedor(nuevoId, "", "", "", "", new Timestamp(System.currentTimeMillis())); if (mostrarFormularioProveedor(nuevoProveedor, true)) {

controlador.agregarProveedor(nuevoProveedor);

}

}

private void editarProveedor() {

int fila = tablaProveedores.getSelectedRow();

if (fila < 0) {

mostrarMensaje("Seleccione un proveedor para editar", 2);

return;

}

String id = (String) modeloTabla.getValueAt(fila, 0);

Proveedor proveedor = controlador.buscarProveedorPorId(id);

if (proveedor != null && mostrarFormularioProveedor(proveedor, false)) {

controlador.editarProveedor(proveedor);

}

}

private void eliminarProveedor() {

int fila = tablaProveedores.getSelectedRow();

if (fila < 0) {

mostrarMensaje("Seleccione un proveedor para eliminar", 2);

return;

}

String id = (String) modeloTabla.getValueAt(fila, 0);

if (confirmarAccion("¿Eliminar este proveedor?")) {

controlador.eliminarProveedor(id);

}

}

private boolean mostrarFormularioProveedor(Proveedor proveedor, boolean esNuevo) {

JDialog dialog = new JDialog(this, esNuevo ? "Nuevo Proveedor" : "Editar Proveedor", true);

dialog.getContentPane().setLayout(new BorderLayout());

dialog.setSize(500, 400); // Aumentamos un poco el tamaño

dialog.setLocationRelativeTo(this);

JPanel formPanel = new JPanel(new GridLayout(6, 2, 10, 10));

formPanel.setBorder(new EmptyBorder(20, 20, 20, 20));

// Campos del formulario

JTextField txtNombre = new JTextField(proveedor.getNombre());

JTextField txtTelefono = new JTextField(proveedor.getTelefono());

JTextField txtDireccion = new JTextField(proveedor.getDireccion());

JTextField txtProducto = new JTextField(proveedor.getProductoSuministrado());

// Configuración del JDateChooser para la fecha

JDateChooser dateChooser = new JDateChooser();

dateChooser.setDateFormatString("dd/MM/yyyy");

// Parsear la fecha actual del proveedor

try {

if (proveedor.getUltimaVisita() != null) {

// Convertir el Timestamp a Date

Date fecha = new Date(proveedor.getUltimaVisita().getTime());

dateChooser.setDate(fecha);

} else {

dateChooser.setDate(new Date()); // Fecha actual por defecto si es null

}

} catch (Exception e) {

System.err.println("Error al establecer fecha: " + e.getMessage());

dateChooser.setDate(new Date()); // Fecha actual por defecto en caso de error

}

// Botón para fecha actual

JButton btnHoy = new JButton("Hoy");

btnHoy.addActionListener(e -> dateChooser.setDate(new Date()));

JPanel fechaPanel = new JPanel(new BorderLayout());

fechaPanel.add(dateChooser, BorderLayout.CENTER);

fechaPanel.add(btnHoy, BorderLayout.EAST);

formPanel.add(new JLabel("Nombre:"));

formPanel.add(txtNombre);

formPanel.add(new JLabel("Teléfono:"));

formPanel.add(txtTelefono);

formPanel.add(new JLabel("Dirección:"));

formPanel.add(txtDireccion);

formPanel.add(new JLabel("Producto:"));

formPanel.add(txtProducto);

formPanel.add(new JLabel("Última Visita:"));

formPanel.add(fechaPanel);

// Botón para registrar visita hoy

JButton btnRegistrarVisita = new JButton("Registrar Visita Hoy");

btnRegistrarVisita.addActionListener(e -> dateChooser.setDate(new Date()));

formPanel.add(btnRegistrarVisita);

JButton btnGuardar = new JButton(esNuevo ? "Registrar" : "Actualizar");

JButton btnCancelar = new JButton("Cancelar");

btnGuardar.addActionListener(e -> {

if (validarCampos(txtNombre, txtTelefono)) {

proveedor.setNombre(txtNombre.getText().trim());

proveedor.setTelefono(txtTelefono.getText().trim());

proveedor.setDireccion(txtDireccion.getText().trim());

proveedor.setProductoSuministrado(txtProducto.getText().trim());

// Obtener la fecha seleccionada y convertir a Timestamp

java.util.Date fechaSeleccionada = dateChooser.getDate();

if (fechaSeleccionada != null) {

proveedor.setUltimaVisita(new Timestamp(fechaSeleccionada.getTime()));

} else {

proveedor.setUltimaVisita(new Timestamp(System.currentTimeMillis()));

}

dialog.dispose();

}

});

btnCancelar.addActionListener(e -> dialog.dispose());

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 10));

buttonPanel.add(btnCancelar);

buttonPanel.add(btnGuardar);

dialog.getContentPane().add(formPanel, BorderLayout.CENTER);

dialog.getContentPane().add(buttonPanel, BorderLayout.SOUTH);

dialog.setVisible(true);

return !proveedor.getNombre().isEmpty();

}

private boolean validarCampos(JTextField nombre, JTextField telefono) {

if (nombre.getText().trim().isEmpty()) {

mostrarMensaje("El nombre es requerido", 3);

return false;

}

if (telefono.getText().trim().isEmpty()) {

mostrarMensaje("El teléfono es requerido", 3);

return false;

}

return true;

}

public void mostrarMensaje(String mensaje, int tipo) {

String titulo = "";

int messageType = JOptionPane.PLAIN\_MESSAGE;

switch (tipo) {

case 1: titulo = "Información"; messageType = JOptionPane.INFORMATION\_MESSAGE; break;

case 2: titulo = "Advertencia"; messageType = JOptionPane.WARNING\_MESSAGE; break;

case 3: titulo = "Error"; messageType = JOptionPane.ERROR\_MESSAGE; break;

}

JOptionPane.showMessageDialog(this, mensaje, titulo, messageType);

}

public boolean confirmarAccion(String mensaje) {

return JOptionPane.showConfirmDialog(

this, mensaje, "Confirmar",

JOptionPane.YES\_NO\_OPTION) == JOptionPane.YES\_OPTION;

}

private void mostrarMenuUsuario() {

JPopupMenu menu = new JPopupMenu();

JMenuItem cambiarUsuario = new JMenuItem("Cambiar usuario");

JMenuItem salir = new JMenuItem("Salir");

cambiarUsuario.addActionListener(e -> {

dispose();

new Login().setVisible(true);

});

salir.addActionListener(e -> System.exit(0));

menu.add(cambiarUsuario);

menu.add(salir);

// Mostrar menú junto al nombre de usuario

Component usuarioLabel = ((JPanel)getContentPane().getComponent(0)).getComponent(0);

menu.show(usuarioLabel, 0, usuarioLabel.getHeight());

}

public JTable getTablaProveedores() {

return this.tablaProveedores;

}

}

Tabla

El contenido generado por IA puede ser incorrecto.

# Usuario

package Controlador;

import Modelo.Usuario;

import java.util.HashMap;

import Modelo.UsuarioDAO;

import java.util.List;

public class GestorUsuario {

private UsuarioDAO usuarioDAO;

public GestorUsuario() {

this.usuarioDAO = new UsuarioDAO();

}

public boolean registrarUsuario(String nombre, String contraseña, String rol) {

return usuarioDAO.registrarUsuario(nombre, contraseña, rol);

}

public boolean autenticar(String username, String password) {

return usuarioDAO.autenticarUsuario(username, password);

}

public List<Usuario> listarUsuarios() {

return usuarioDAO.obtenerTodosUsuarios();

}

public boolean cambiarRolUsuario(String username, String nuevoRol) {

return usuarioDAO.actualizarRol(username, nuevoRol);

}

public boolean eliminarUsuario(String username) {

return usuarioDAO.eliminarUsuario(username);

}

}

package Modelo;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import ConexionBD.ConexionAccess;

public class UsuarioDAO {

private Connection conn;

public UsuarioDAO() {

// Inicializar conexión

conn = ConexionAccess.conectar();

crearTablaSiNoExiste();

}

private void crearTablaSiNoExiste() {

try {

// Verificar si la tabla existe

boolean tablaExiste = false;

DatabaseMetaData meta = conn.getMetaData();

try (ResultSet tables = meta.getTables(null, null, "Usuarios", new String[]{"TABLE"})) {

tablaExiste = tables.next();

}

if (!tablaExiste) {

try (Statement stmt = conn.createStatement()) {

// Crear tabla

stmt.execute("CREATE TABLE Usuarios (" +

"username VARCHAR(50) PRIMARY KEY, " +

"password VARCHAR(50) NOT NULL, " +

"rol VARCHAR(20) NOT NULL)");

// Insertar admin por defecto

stmt.execute("INSERT INTO Usuarios (username, password, rol) VALUES " +

"('admin', 'admin123', 'Admin')");

System.out.println("Tabla Usuarios creada e inicializada");

}

}

} catch (SQLException e) {

System.err.println("Error crítico al inicializar tabla Usuarios: ");

e.printStackTrace();

// Puedes agregar aquí un JOptionPane.showMessageDialog para informar al usuario

throw new RuntimeException("No se pudo inicializar la tabla Usuarios", e);

}

}

private boolean existeUsuario(String username) {

String sql = "SELECT username FROM Usuarios WHERE username = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, username);

return pstmt.executeQuery().next();

} catch (SQLException e) {

return false;

}

}

public boolean autenticarUsuario(String username, String password) {

String sql = "SELECT \* FROM Usuarios WHERE username = ? AND password = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, username);

pstmt.setString(2, password);

return pstmt.executeQuery().next();

} catch (SQLException e) {

return false;

}

}

public boolean registrarUsuario(String username, String password, String rol) {

String sql = "INSERT INTO Usuarios (username, password, rol) VALUES (?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, username);

pstmt.setString(2, password);

pstmt.setString(3, rol);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

return false;

}

}

public Usuario obtenerUsuario(String username) {

String sql = "SELECT \* FROM Usuarios WHERE username = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, username);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return new Usuario(

rs.getString("username"),

rs.getString("password"),

rs.getString("rol")

);

}

} catch (SQLException e) {

System.err.println("Error al obtener usuario: " + e.getMessage());

}

return null;

}

public List<Usuario> obtenerTodosUsuarios() {

List<Usuario> usuarios = new ArrayList<>();

String sql = "SELECT \* FROM Usuarios";

try (Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(sql)) {

while (rs.next()) {

usuarios.add(new Usuario(

rs.getString("username"),

rs.getString("password"),

rs.getString("rol")

));

}

} catch (SQLException e) {

System.err.println("Error al obtener usuarios: " + e.getMessage());

}

return usuarios;

}

public boolean actualizarRol(String username, String nuevoRol) {

String sql = "UPDATE Usuarios SET rol = ? WHERE username = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, nuevoRol);

pstmt.setString(2, username);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

return false;

}

}

public boolean eliminarUsuario(String username) {

String sql = "DELETE FROM Usuarios WHERE username = ? AND username <> 'admin'";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, username);

return pstmt.executeUpdate() > 0;

} catch (SQLException e) {

return false;

}

}

public String obtenerRolUsuario(String username) {

String sql = "SELECT rol FROM Usuarios WHERE username = ?";

try (PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, username);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

return rs.getString("rol");

}

} catch (SQLException e) {

System.err.println("Error al obtener rol: " + e.getMessage());

}

return null; // Retorna null si no encuentra el usuario o hay error

}

}

package Vista;

import Modelo.Usuario;

import Modelo.UsuarioDAO;

import Controlador.GestorUsuario;

import Controlador.ReportesControlador;

import javax.swing.\*;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.border.\*;

import java.util.List;

public class gestionUsuario extends JFrame {

private Usuario usuario;

private GestorUsuario gestorUsuario;

private JTable tablaUsuarios;

private DefaultTableModel modeloTabla;

private JButton btnAgregar, btnEditar, btnEliminar;

public gestionUsuario(Usuario usuario) {

this.usuario = usuario;

this.gestorUsuario = new GestorUsuario();

this.modeloTabla = new DefaultTableModel(new Object[]{"Usuario", "Rol", "Estado"}, 0);

initUI();

cargarUsuarios();

}

private void initUI() {

setTitle("El Habanerito - Gestión de Usuarios");

setSize(1200, 800);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

// Panel principal con BorderLayout

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.setBorder(new EmptyBorder(0, 0, 0, 0));

mainPanel.setBackground(new Color(240, 240, 240));

// 1. Panel superior (logo + usuario)

JPanel topPanel = crearPanelSuperior();

// 2. Menú horizontal

JPanel menuPanel = crearMenuHorizontal();

// 3. Panel para la tabla (usamos un panel contenedor)

JPanel tablePanel = new JPanel(new BorderLayout());

tablaUsuarios = new JTable(modeloTabla);

configurarTabla();

JScrollPane scrollPane = new JScrollPane(tablaUsuarios);

scrollPane.setBorder(BorderFactory.createCompoundBorder(

new EmptyBorder(15, 15, 15, 15),

BorderFactory.createLineBorder(new Color(200, 200, 200), 1, true)

));

tablePanel.add(scrollPane, BorderLayout.CENTER);

// 4. Panel inferior rosa con botones

JPanel bottomPanel = crearPanelInferior();

// Panel contenedor para menú y tabla

JPanel centerContainer = new JPanel(new BorderLayout());

centerContainer.add(menuPanel, BorderLayout.NORTH);

centerContainer.add(tablePanel, BorderLayout.CENTER);

// Ensamblar componentes

mainPanel.add(topPanel, BorderLayout.NORTH);

mainPanel.add(centerContainer, BorderLayout.CENTER);

mainPanel.add(bottomPanel, BorderLayout.SOUTH);

getContentPane().add(mainPanel);

}

// ==================== MÉTODOS DE DISEÑO (iguales a proveedores.java) ====================

private JPanel crearPanelSuperior() {

JPanel topPanel = new JPanel(new BorderLayout());

topPanel.setBackground(new Color(255, 198, 144));

topPanel.setPreferredSize(new Dimension(0, 60));

topPanel.setBorder(BorderFactory.createEmptyBorder(5, 15, 5, 15));

JPanel logoUserPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 5, 0));

logoUserPanel.setOpaque(false);

try {

ImageIcon originalIcon = new ImageIcon("imagen\\logo.png");

Image resizedImage = originalIcon.getImage().getScaledInstance(60, 60, Image.SCALE\_SMOOTH);

JLabel logo = new JLabel(new ImageIcon(resizedImage));

logoUserPanel.add(logo);

} catch (Exception e) {

logoUserPanel.add(new JLabel("Logo"));

}

JLabel lblUsuario = new JLabel(usuario.getUsername());

lblUsuario.setFont(new Font("Arial", Font.BOLD, 14));

lblUsuario.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

lblUsuario.addMouseListener(new MouseAdapter() {

public void mouseClicked(MouseEvent e) {

mostrarMenuUsuario();

}

});

logoUserPanel.add(Box.createRigidArea(new Dimension(5, 0)));

logoUserPanel.add(lblUsuario);

JPanel containerPanel = new JPanel(new BorderLayout());

containerPanel.setOpaque(false);

containerPanel.add(logoUserPanel, BorderLayout.EAST);

topPanel.add(containerPanel, BorderLayout.CENTER);

return topPanel;

}

private JPanel crearMenuHorizontal() {

JPanel menuPanel = new JPanel(new GridBagLayout()); // Cambiamos a GridBagLayout

menuPanel.setBackground(new Color(230, 230, 230));

menuPanel.setPreferredSize(new Dimension(getWidth(), 50)); // Altura fija de 50px

menuPanel.setBorder(BorderFactory.createMatteBorder(1, 0, 1, 0, Color.GRAY));

String[] opciones = {"Productos", "Reportes", "Inventario", "Cliente", "Proveedores", "Usuarios", "Salir"};

GridBagConstraints gbc = new GridBagConstraints();

gbc.fill = GridBagConstraints.BOTH;

gbc.weightx = 1.0; // Distribuye el espacio equitativamente

gbc.weighty = 1.0;

for (String opcion : opciones) {

JButton btn = crearBotonMenu(opcion, opcion.equals("Usuarios"));

btn.addActionListener(e -> manejarAccionMenu(opcion));

// Configuración específica para cada botón

gbc.gridwidth = 1; // Cada botón ocupa 1 celda

menuPanel.add(btn, gbc);

}

return menuPanel;

}

private JButton crearBotonMenu(String texto, boolean esActivo) {

JButton boton = new JButton(texto);

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setFocusPainted(false);

boton.setBorder(BorderFactory.createEmptyBorder(10, 5, 10, 5)); // Padding interno reducido

// Estilo para el estado normal

if (esActivo) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.BLACK);

} else {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

// Efecto hover

boton.addMouseListener(new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

if (!esActivo) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

}

@Override

public void mouseExited(MouseEvent e) {

if (!esActivo) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

}

});

// Tamaño preferido más compacto

boton.setPreferredSize(new Dimension(100, 40)); // Ancho mínimo sugerido

return boton;

}

private JPanel crearPanelInferior() {

JPanel panel = new JPanel(new BorderLayout());

panel.setPreferredSize(new Dimension(0, 100));

panel.setBackground(new Color(255, 182, 193));

panel.setBorder(BorderFactory.createMatteBorder(1, 0, 0, 0, Color.GRAY));

JPanel buttonPanel = new JPanel();

buttonPanel.setOpaque(false);

btnAgregar = crearBotonAccion("Agregar");

btnEditar = crearBotonAccion("Editar");

btnEliminar = crearBotonAccion("Eliminar");

btnAgregar.addActionListener(e -> agregarUsuario());

btnEditar.addActionListener(e -> editarUsuario());

btnEliminar.addActionListener(e -> eliminarUsuario());

buttonPanel.add(btnAgregar);

buttonPanel.add(Box.createRigidArea(new Dimension(20, 0)));

buttonPanel.add(btnEditar);

buttonPanel.add(Box.createRigidArea(new Dimension(20, 0)));

buttonPanel.add(btnEliminar);

panel.add(buttonPanel, BorderLayout.CENTER);

return panel;

}

private JButton crearBotonAccion(String texto) {

JButton boton = new JButton(texto) {

@Override

protected void paintBorder(Graphics g) {

Graphics2D g2 = (Graphics2D) g.create();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.setColor(Color.DARK\_GRAY);

g2.drawRoundRect(0, 0, getWidth()-1, getHeight()-1, 15, 15);

g2.dispose();

}

};

boton.setFont(new Font("Arial", Font.BOLD, 14));

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

boton.setFocusPainted(false);

boton.setContentAreaFilled(false);

boton.setOpaque(true);

boton.setPreferredSize(new Dimension(255, 60));

boton.setBorder(BorderFactory.createEmptyBorder(5, 20, 5, 20));

boton.addMouseListener(new MouseAdapter() {

public void mouseEntered(MouseEvent e) {

boton.setBackground(new Color(216, 237, 88));

boton.setForeground(Color.WHITE);

}

public void mouseExited(MouseEvent e) {

boton.setBackground(Color.GRAY);

boton.setForeground(Color.BLACK);

}

});

return boton;

}

private void configurarTabla() {

tablaUsuarios.setRowHeight(30);

tablaUsuarios.setFont(new Font("Arial", Font.PLAIN, 14));

tablaUsuarios.getTableHeader().setFont(new Font("Arial", Font.BOLD, 14));

tablaUsuarios.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();

centerRenderer.setHorizontalAlignment(JLabel.CENTER);

for (int i = 0; i < tablaUsuarios.getColumnCount(); i++) {

tablaUsuarios.getColumnModel().getColumn(i).setCellRenderer(centerRenderer);

}

tablaUsuarios.setDefaultRenderer(Object.class, new DefaultTableCellRenderer() {

@Override

public Component getTableCellRendererComponent(JTable table, Object value,

boolean isSelected, boolean hasFocus, int row, int column) {

JLabel label = (JLabel) super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

label.setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));

if (isSelected) {

label.setBackground(new Color(216, 237, 88));

label.setForeground(Color.BLACK);

} else {

label.setBackground(row % 2 == 0 ? Color.WHITE : new Color(240, 240, 240));

}

return label;

}

});

}

// ==================== MÉTODOS DE GESTIÓN ====================

private void cargarUsuarios() {

// Limpiar la tabla

modeloTabla.setRowCount(0);

// Obtener todos los usuarios del gestor

List<Usuario> usuarios = gestorUsuario.listarUsuarios();

// Agregar cada usuario a la tabla

for (Usuario usuario : usuarios) {

modeloTabla.addRow(new Object[]{

usuario.getUsername(),

usuario.getRol(),

"Activo" // Estado por defecto

});

}

}

private void agregarUsuario() {

JDialog dialog = new JDialog(this, "Nuevo Usuario", true);

dialog.setSize(400, 250);

dialog.setLocationRelativeTo(this);

JPanel formPanel = new JPanel(new GridLayout(3, 2, 10, 10));

formPanel.setBorder(new EmptyBorder(20, 20, 20, 20));

JTextField txtUsername = new JTextField();

JPasswordField txtPassword = new JPasswordField();

JComboBox<String> comboRol = new JComboBox<>(new String[]{"Admin", "Trab"});

formPanel.add(new JLabel("Usuario:"));

formPanel.add(txtUsername);

formPanel.add(new JLabel("Contraseña:"));

formPanel.add(txtPassword);

formPanel.add(new JLabel("Rol:"));

formPanel.add(comboRol);

JButton btnGuardar = new JButton("Guardar");

JButton btnCancelar = new JButton("Cancelar");

btnGuardar.addActionListener(e -> {

String username = txtUsername.getText().trim();

String password = new String(txtPassword.getPassword()).trim();

String rol = (String) comboRol.getSelectedItem();

if (username.isEmpty() || password.isEmpty()) {

JOptionPane.showMessageDialog(dialog,

"Complete todos los campos",

"Error",

JOptionPane.ERROR\_MESSAGE);

return;

}

// Intenta registrar el usuario

if (gestorUsuario.registrarUsuario(username, password, rol)) {

// Actualiza la tabla con los nuevos datos

cargarUsuarios();

JOptionPane.showMessageDialog(dialog,

"Usuario registrado exitosamente",

"Éxito",

JOptionPane.INFORMATION\_MESSAGE);

dialog.dispose();

} else {

JOptionPane.showMessageDialog(dialog,

"El usuario ya existe",

"Error",

JOptionPane.ERROR\_MESSAGE);

}

});

btnCancelar.addActionListener(e -> dialog.dispose());

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 10));

buttonPanel.add(btnCancelar);

buttonPanel.add(btnGuardar);

dialog.getContentPane().add(formPanel, BorderLayout.CENTER);

dialog.getContentPane().add(buttonPanel, BorderLayout.SOUTH);

dialog.setVisible(true);

}

private void editarUsuario() {

int fila = tablaUsuarios.getSelectedRow();

if (fila < 0) {

JOptionPane.showMessageDialog(this, "Seleccione un usuario", "Advertencia", JOptionPane.WARNING\_MESSAGE);

return;

}

String username = (String) modeloTabla.getValueAt(fila, 0);

String rolActual = (String) modeloTabla.getValueAt(fila, 1);

String nuevoRol = (String) JOptionPane.showInputDialog(

this,

"Seleccione el nuevo rol:",

"Editar Rol",

JOptionPane.PLAIN\_MESSAGE,

null,

new String[]{"Admin", "Trab"},

rolActual

);

if (nuevoRol != null && !nuevoRol.equals(rolActual)) {

if (gestorUsuario.cambiarRolUsuario(username, nuevoRol)) {

cargarUsuarios();

} else {

JOptionPane.showMessageDialog(this, "No se pudo actualizar el rol", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

private void eliminarUsuario() {

int fila = tablaUsuarios.getSelectedRow();

if (fila < 0) {

JOptionPane.showMessageDialog(this, "Seleccione un usuario", "Advertencia", JOptionPane.WARNING\_MESSAGE);

return;

}

String username = (String) modeloTabla.getValueAt(fila, 0);

if (username.equals("admin")) {

JOptionPane.showMessageDialog(this, "No se puede eliminar al administrador", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int confirm = JOptionPane.showConfirmDialog(

this,

"¿Eliminar al usuario " + username + "?",

"Confirmar",

JOptionPane.YES\_NO\_OPTION

);

if (confirm == JOptionPane.YES\_OPTION) {

if (gestorUsuario.eliminarUsuario(username)) {

cargarUsuarios();

} else {

JOptionPane.showMessageDialog(this, "No se pudo eliminar el usuario", "Error", JOptionPane.ERROR\_MESSAGE);

}

}

}

// ==================== MÉTODOS DE NAVEGACIÓN ====================

private void manejarAccionMenu(String opcion) {

switch (opcion) {

case "Salir":

this.dispose();

new menuprincipal(usuario).setVisible(true);

break;

case "Productos":

this.dispose();

new producto(usuario).setVisible(true);

break;

case "Reportes":

this.dispose();

reportes vistaReportes = new reportes(usuario, new ReportesControlador(null, usuario));

vistaReportes.setVisible(true); // Muestra la ventana

break;

case "Inventario":

this.dispose();

new inventario(usuario).setVisible(true);

break;

case "Cliente":

this.dispose();

new clientes(usuario, null).setVisible(true);

break;

case "Proveedores":

this.dispose();

new proveedores(usuario).setVisible(true);

break;

case "Usuarios":

JOptionPane.showMessageDialog(this, "Ya estás en la ventana de Usuario.");

break;

}

}

private void mostrarMenuUsuario() {

JPopupMenu menu = new JPopupMenu();

JMenuItem cambiarUsuario = new JMenuItem("Cambiar usuario");

JMenuItem salir = new JMenuItem("Salir");

cambiarUsuario.addActionListener(e -> {

dispose();

new Login().setVisible(true);

});

salir.addActionListener(e -> System.exit(0));

menu.add(cambiarUsuario);

menu.add(salir);

Component usuarioLabel = ((JPanel)getContentPane().getComponent(0)).getComponent(0);

menu.show(usuarioLabel, 0, usuarioLabel.getHeight());

}

}

Texto

El contenido generado por IA puede ser incorrecto.