

**FACULTAD DE INGENIERÍA**

**Carrera de Ingeniería de Software**

TRABAJO FINAL

**CURSO:**

**PROGRAMACIÓN ORIENTADA A OBJETOS I**

**BLOQUE: FC-PREISF02A1M**

**DOCENTE:**

DIAZ ARENAS, DANIEL JESUS

**GRUPO:**

RIVAS PERALES, ERNESTO SEBASTIAN

BRAVO OCAMPO, DAVID AQUILES

**Lima – Perú 2023**

# TABLA DE CONTENIDO

[TABLA DE CONTENIDO 2](#_Toc127781206)

[INTRODUCCIÓN 3](#_Toc127781207)

[DESCRIPCIÓN DEL PROBLEMA COMPUTACIONAL 3](#_Toc127781208)

[DESCRIPCIÓN DEL CASO DE ESTUDIO 3](#_Toc127781209)

[OBJETIVO GENERAL 4](#_Toc127781210)

[OBJETIVOS ESPECÍFICOS 4](#_Toc127781211)

[DESARROLLO DEL APLICATIVO 4](#_Toc127781212)

[1. DISEÑO DE LA INTERFAZ O EL USO DE LA CONSOLA 4](#_Toc127781213)

[2. DESARROLLO DEL PRODUCTO 26](#_Toc127781214)

[VIEW/ 26](#_Toc127781215)

[Main.cpp 27](#_Toc127781216)

[OptionsMenu.h 27](#_Toc127781217)

[Funciones.h 27](#_Toc127781218)

[MODEL/ 26](#_Toc127781219)

[CONTROLLER/ 26](#_Toc127781220)

[CONCLUSIONES 26](#_Toc127781221)

[GLOSARIO DE TÉRMINOS 28](#_Toc127781222)

[REFERENCIAS 28](#_Toc127781223)

# INTRODUCCIÓN

Este informe presenta la creación de un programa en C++ para ser utilizado en una farmacia ficticia, en este caso llamada “Farmacia ‘Miriam’”. El objetivo del proyecto es proporcionar una herramienta informática que permita a los empleados de este negocio gestionar el Kárdex1 de la empresa, pasando por las ventas, las compras de nuevos productos, datos del inventario y cada uno de los productos, y funcionalidades adicionales de seguridad y soporte.

El programa cuenta con una interfaz de usuario basada en texto. Múltiples menús de opciones, listado y pantallas estandarizadas para la comodidad del usuario. Adicionalmente se propicia un manejo de los datos por medio de archivos guardados de manera local.

En este informe, se describe el proceso de diseño, desarrollo e implementación del programa, incluyendo un manual de usuario y la implementación del código.

El programa desarrollado se espera que sea adaptable, altamente modificable y escalable. Apunta a ser un programa que posea uso práctico en un caso real y ayude a reducir la carga de trabajo que conlleva una cuenta manual.

# DESCRIPCIÓN DEL PROBLEMA COMPUTACIONAL

El problema computacional que se aborda en este proyecto es el llevar registro de entradas y salidas de la mercadería. Inventarios actuales y un registro básico de los clientes. Aligerar la carga de manera efectiva es una de las principales necesidades.

# DESCRIPCIÓN DEL CASO DE ESTUDIO

El caso de estudio que se aborda en este proyecto se refiere a una farmacia ficticia llamada “Miriam”, ubicada en una zona comercial de tamaño medio. La farmacia “Miriam” tiene un promedio de cien clientes por día, con un pico de ventas durante el fin de semana. Los clientes compran una variedad de productos, incluyendo medicamentos de venta libre y recetados, suplementos dietéticos, productos de cuidado personal, entre otros.

Se lleva a cabo la gestión de inventario y ventas manualmente, lo que puede llevar mucho tiempo y ser propenso a errores. Además, el personal de la farmacia a menudo tiene dificultades para encontrar información sobre un producto en particular, lo que puede retrasar las ventas y afectar la satisfacción del cliente.

No se lleva un registro exacto de las entradas y salidas del día en la caja. Así como el monto que se retira de esta, quien lo hizo y cuando se hizo. Añadimos que otro problema común es el conteo de productos y la disponibilidad de estos.

Debido a esto, se ha planteado la necesidad de desarrollar un programa en C++ para solucionar los desafíos actuales de ventas y el inventario en la farmacia.

# OBJETIVO GENERAL

Un sistema que posea inicio de sesión y múltiples niveles de usuario (mínimo 3 correspondiente al vendedor, supervisor y administrador), así como el registro de nuevos usuarios. Un sistema que permita consultar inventario y estado de caja, registrar compras y ventas así como nuevos clientes y proveedores. Por último el registro de movimientos independientes es indispensable para registrar bajas del inventario por merma, salida operativa, entrada operativa, etc.

# OBJETIVOS ESPECÍFICOS

* Sistema de Inicio de Sesión y Registro con niveles de usuario.
* Sistema de Menús cómodos.
* Kárdex1 que registre las ventas, compras y otros movimientos.
* Registros independientes (a parte del Kárdex1) de las ventas y compras
* Registros de los detalles de venta y compra.
* Una barrera de inicio de sesión para poder acceder al sistema.
* Sistema de visualización del inventario y datos de los productos.

# DESARROLLO DEL APLICATIVO

## DISEÑO DE LA INTERFAZ O EL USO DE LA CONSOLA

No se incluyen las pantallas de error (autodescriptivas)

**Menú de Inicio** (en caso de no tener administradores registrados pasa directamente a registrar uno nuevo)

Texto

Descripción generada automáticamente

**Inicio de Sesión**

Texto

Descripción generada automáticamente

**Menú Principal**

Posee las opciones de Registrar nueva Venta, entrar al menú de Usuario, entrar al Menú de Inventario, entrar al Menú de ‘Otros’ y salir del programa respectivamente.

Texto

Descripción generada automáticamente

**Registrar Venta**



En caso de desear otra fecha:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opción 1:

Texto

Descripción generada automáticamente

Opción 2:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

En caso el cliente no esté registrado:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

[Volvemos a la Opción 2](#_DISEÑO_DE_LA)

Texto

Descripción generada automáticamente

Opción 1:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opción 2: (listado con los productos en el carrito)

Texto

Descripción generada automáticamente

Opción 3: (listado con los productos en el carrito)

Texto

Descripción generada automáticamente

Opción 4:

Texto

Descripción generada automáticamente

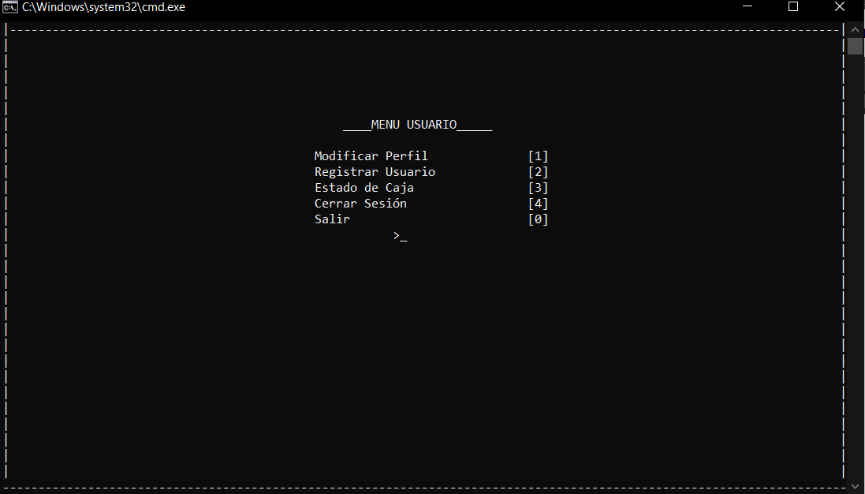
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Menú de Usuario:**



Opción 1:

Texto

Descripción generada automáticamente

Todas las opciones son similares (1-5)

Texto

Descripción generada automáticamente

Opción 6:

Texto

Descripción generada automáticamente

Opción 2:

Si el usuario intenta crear un perfil necesitará ser para vendedor, supervisor o administrador; para supervisor, administrador y para administrador, volver a iniciar sesión.



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opción 3:

Texto

Descripción generada automáticamente

**Menú de Inventario:**

Texto

Descripción generada automáticamente

Opción 1: (Bastante ya explicado en “Registrar Venta”)

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opción 1:

Texto

Descripción generada automáticamente

Opción 2:

Texto

Descripción generada automáticamente

En caso el Proveedor no esté Registrado:

Texto

Descripción generada automáticamente

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

[Volvemos a la Opción 2](#_DISEÑO_DE_LA)

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opción 1:

Texto

Descripción generada automáticamente



Texto

Descripción generada automáticamente

Opción 2: (listado con los productos en el carrito)

Texto

Descripción generada automáticamente

Opción 3: (listado con los productos en el carrito)

Texto

Descripción generada automáticamente

Opción 4:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opción 2:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opción 3:

Texto

Descripción generada automáticamente

Opción 1:

Texto

Descripción generada automáticamente

Opción 2:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Todas las opciones son similares (1-4)

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Opciones 4 - 5 son similares

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## DESARROLLO DEL PRODUCTO

#### VIEW/

##### Main.cpp

#include <iostream>

#include <string>

#include <vector>

#include <random>

#include <ctime>

#include <unistd.h>

#include "../model/Usuario.cpp"

#include "../model/models.h"

#include "../controller/controllers.h"

#include "OptionMenus.h"

using namespace std;

/\*

1) INICIO DE SESIÓN

    \*>REGISTRARSE

    >INICIAR SESION

2) MENU PRINCIPAL

    >REGISTRAR VENTA

    >/[USUARIO]/

    >INVENTARIO (MODIFICAR Y CONSULTAR)

    >REGISTROS (PERSONALES{DIA, GENERAL} O OTRO USUARIO{SUPERVISOR PARA REVISAR DE VENDEDORES Y ADM PARA REVISAR DE SUPERVISORES} Y VENTAS)

3) USUARIO

    >MODIFICAR

    >REGISTROS

    \*>REGISTRARSE

    >ESTADO DE CAJA

    >CERRAR SESION

4) INVENTARIO

    >AÑADIR

    >CONSULTAR

    >MODIFICAR

5) REGISTROS

    >ESTADO DE CAJA

    >VENTAS PERSONALES

    >CONSULTAR OTROS

\*/

int main(int argc, char \*argv[])

{

    SetConsoleCP(1252);

    SetConsoleOutputCP(1252);

    srand(time(NULL));

    if (userController.existeAdministrador())

    {

        // progController.openSesion(0, true, true);

        if (!progController.getActiveSesion())

            menuLogIn();

        else

            menuMain();

    }

    else

        doRegistrarse(true);

    esquinarTexto("...", 2, false, true, true);

    return 0;

}

void doRegistrarse(bool start = false)

{

    string type;

    string username;

    string nombre;

    string apellidos;

    string strInput;

    long int numDocumento;

    string tipoUsuario;

    string tipoDocumento;

    string contrasena;

    string contrasenaConfi;

    int docSize;

    bool loged = false;

    int opt;

    vector<string> options;

    int sizeY = 3, e = 0;

    system("cls");

    if (!start)

    {

        options = {"VENDEDOR", "SUPERVISOR", "ADMINISTRADOR"};

        opt = menu("REGISTRO - TIPO DE USUARIO", options);

        switch (opt)

        {

        case 1:

            if (!progController.getActiveSesion() || !progController.getIsSupervisor())

            {

                menuListado({"~-NECESITAS INICIAR SESIÓN COMO SUPERVISOR O ADMINISTRADOR-~"}, 0, "", false);

                Sleep(1000);

                loged = doIniciarSesion(false, type);

                if (loged && (type == "Administrador" || type == "Supervisor"))

                    tipoUsuario = "Vendedor";

                else

                {

                    menuListado({"~-CANCELASTE EL REGISTRO-~"}, 0, "", true);

                    tipoUsuario = "Cancelar";

                    esquinarTexto("", 1, false, true, false);

                    system("pause");

                }

            }

            else

                tipoUsuario = "Vendedor";

            break;

        case 2:

            if (!progController.getActiveSesion() || !progController.getIsAdmin())

            {

                menuListado({"~-NECESITAS INICIAR SESIÓN COMO ADMINISTRADOR-~"}, 0, "", false);

                Sleep(1000);

                loged = doIniciarSesion(false, type);

                if (loged && type == "Administrador")

                    tipoUsuario = "Supervisor";

                else

                {

                    menuListado({"~-CANCELASTE EL REGISTRO-~"}, 0, "", true);

                    tipoUsuario = "Cancelar";

                    esquinarTexto("", 1, false, true, false);

                    system("pause");

                }

            }

            else

                tipoUsuario = "Supervisor";

            break;

        case 3:

            menuListado({"~-NECESITAS INICIAR SESIÓN / VOLVER A INICIAR COMO ADMINISTRADOR-~"}, 0, "", false);

            Sleep(1000);

            loged = doIniciarSesion(false, type);

            if (loged && type == "Administrador")

                tipoUsuario = "Administrador";

            else

            {

                menuListado({"~-CANCELASTE EL REGISTRO-~"}, 0, "", true);

                tipoUsuario = "Cancelar";

                esquinarTexto("", 1, false, true, false);

                system("pause");

            }

            break;

        case 0:

            tipoUsuario = "Cancelar";

            break;

        };

        if (tipoUsuario == "Cancelar")

            return;

    }

    else

    {

        menuError({"No tenemos un administrador registrado", "REGISTRA UNO NUEVO"});

        tipoUsuario = "Administrador";

    }

    vector<string> inputs;

    do

    {

        inputs.clear();

        options = {"Nombre de usuario", "Nombres", "Apellidos"};

        menuDatos(options, inputs, 0, 0, "\_\_DATOS\_\_");

        username = inputs[0].substr(0, inputs[0].find(' '));

        nombre = inputs[1];

        apellidos = inputs[2];

        if (userController.existeUsuario(username))

            menuError({"~USUARIO YA EXISTE, ESCOGE OTRO~"});

    } while (userController.existeUsuario(username));

    options = {"DNI", "CARNET EXTRANJERIA (CE)", "PASAPORTE", "RUC"}; // Carritos de compra, etc

    opt = menu("REGISTRO - TIPO DE USUARIO", options);

    switch (opt)

    {

    case 1:

        tipoDocumento = "DNI";

        docSize = 8;

        break;

    case 2:

        tipoDocumento = "CE";

        docSize = 12;

        break;

    case 3:

        tipoDocumento = "PASAPORTE";

        docSize = 12;

        break;

    case 4:

        tipoDocumento = "RUC";

        docSize = 11;

        break;

    case 0:

        tipoDocumento = "Cancelar";

        break;

    }

    if (tipoDocumento == "Cancelar")

        return;

    strInput = "0";

    do

    {

        inputs.clear();

        menuDatos({"Numero de documento"}, inputs, 0, 0, "\_" + tipoDocumento + "\_ -('Salir' para salir)");

        strInput = inputs[0].substr(0, inputs[0].find(' '));

        if (aMinuscula(strInput) == "salir")

            return;

        if ((int)strInput.size() != docSize)

        {

            options = {"~-Los documentos tipo " + tipoDocumento + " deben contener " + to\_string(docSize) + " digitos-~", "~VUELVA A INGRESAR SU NUMERO DE DOCUMENTO~"};

            menuError(options, 1);

        }

        if (!esNumero(strInput))

            menuError({"~[[INTRODUCE UN VALOR NUMERICO]]~"});

    } while (!esNumero(strInput) || (int)strInput.size() != docSize);

    numDocumento = stoll(strInput);

    int sizePass;

    do

    {

        ;

        inputs.clear();

        menuDatos({"Contrasena", "Repetir Contrasena"}, inputs, 2, 1);

        contrasena = sha256(inputs[0]);

        contrasenaConfi = sha256(inputs[1]);

        sizePass = inputs[0].size();

        inputs.clear();

        if (sizePass < 8)

            menuError({"~-La contrasena debe tener minimo 8 caracteres-~", "~VUELVA A INGRESAR UNA CONTRASENA~"});

        else if (contrasena != contrasenaConfi)

            menuError({"~-Las contrasenas no son iguales-~", "~VUELVA A INGRESAR UNA CONTRASENA~"});

    } while (contrasena != contrasenaConfi || sizePass < 8);

    Usuario objUser(userController.getNewCodUsuario(), username, contrasena, nombre, apellidos, tipoDocumento, numDocumento, tipoUsuario);

    userController.add(objUser);

    objUser.listarDatos(); // BORRAR VERSION FINAL - SOLO DEBUG

    system("cls");

    userController.archGrabarDatos();

    if (start)

    {

        if (!doIniciarSesion(true, type))

            menuLogIn();

    }

}

void doModificarPerfil(int key)

{

    string temporal, temporal2;

    int opt, sizeTemporal;

    Usuario objUser = userController.getUsuario(key);

    vector<string> options;

    vector<string> inputs;

    do

    {

        inputs.clear();

        options = {"Usuario", "Nombres", "Apellidos", "Contrasena", "Documento", "Guardar Cambios"};

        opt = menu("EDITAR PERFIL", options);

        switch (opt)

        {

        case 1:

            menuDatos({"Nombre de usuario"}, inputs);

            temporal = inputs[0].substr(0, inputs[0].find(' '));

            if (userController.existeUsuario(temporal) && temporal != objUser.getUsername())

                menuError({"##[USUARIO YA EXISTE, ESCOGE OTRO NOMBRE]##"});

            if (menuConfirmar("Desea que\"" + temporal + "\"sea su nuevo nombre de usuario"))

                objUser.setUsername(temporal);

            break;

        case 2:

            menuDatos({"Nombre"}, inputs);

            temporal = inputs[0];

            if (menuConfirmar("Desea que\"" + temporal + "\"sea su nuevo nombre"))

                objUser.setNombre(temporal);

            break;

        case 3:

            menuDatos({"Apellidos"}, inputs);

            temporal = inputs[0];

            if (menuConfirmar("Desea que\"" + temporal + "\"sea sus nuevos apellidos"))

                objUser.setApellidos(temporal);

            break;

        case 4:

            do

            {

                inputs.clear();

                menuDatos({"Contrasena", "Repetir Contrasena"}, inputs, 2, 1, "-('Salir' para salir)-");

                if (aMinuscula(inputs[0]) != "salir" && aMinuscula(inputs[1]) != "salir")

                {

                    temporal = sha256(inputs[0]);

                    temporal2 = sha256(inputs[1]);

                    sizeTemporal = inputs[0].size();

                    inputs.clear();

                }

                else

                    break;

                if (sizeTemporal < 8)

                    menuError({"~-La contrasena debe tener minimo 8 caracteres-~", "~VUELVA A INGRESAR UNA CONTRASENA~"});

                else if (temporal != temporal2)

                    menuError({"~-Las contrasenas no son iguales-~", "~VUELVA A INGRESAR UNA CONTRASENA~"});

            } while (temporal != temporal2 || sizeTemporal < 8);

            objUser.setContrasena(temporal);

            temporal = temporal2 = "";

            sizeTemporal = 0;

            break;

        case 5:

            options = {"DNI", "CARNET EXTRANJERIA (CE)", "PASAPORTE", "RUC"}; // Carritos de compra, etc

            opt = menu("REGISTRO - TIPO DE USUARIO", options);

            switch (opt)

            {

            case 1:

                temporal = "DNI";

                sizeTemporal = 8;

                break;

            case 2:

                temporal = "CE";

                sizeTemporal = 12;

                break;

            case 3:

                temporal = "PASAPORTE";

                sizeTemporal = 12;

                break;

            case 4:

                temporal = "RUC";

                sizeTemporal = 11;

                break;

            case 0:

                temporal = "Cancelar";

                break;

            }

            if (temporal == "Cancelar")

                break;

            temporal2 = "0";

            do

            {

                inputs.clear();

                menuDatos({"Numero de documento"}, inputs, 0, 0, "\_" + temporal + "\_ -('Salir' para salir)");

                temporal2 = inputs[0].substr(0, inputs[0].find(' '));

                if (aMinuscula(temporal2) == "salir")

                    return;

                if ((int)temporal2.size() != sizeTemporal)

                {

                    options = {"~-Los documentos tipo " + temporal + " deben contener " + to\_string(sizeTemporal) + " digitos-~", "~VUELVA A INGRESAR SU NUMERO DE DOCUMENTO~"};

                    menuError(options, 1);

                }

                if (!esNumero(temporal2))

                    menuError({"~[[INTRODUCE UN VALOR NUMERICO]]~"});

            } while (!esNumero(temporal2) || (int)temporal2.size() != sizeTemporal);

            objUser.setDocumento(temporal);

            objUser.setNumDocumento(stoll(temporal2));

            break;

        case 6:

            if (menuConfirmar("Desea guardar los cambios", "Deberas reiniciar la sesion"))

            {

                userController.modify(objUser, key);

                userController.archGrabarDatos();

                progController.closeSesion();

            }

            else

                menuError({"Cancelando los cambios..."});

            break;

        case 0:

            if (menuConfirmar("Desea cancelar los cambios", "Se perderAn permanentemente"))

                menuError({"Cancelando los cambios..."});

            else

                opt = 7;

            break;

        }

    } while (opt != 6 && opt != 0);

}

bool doIniciarSesion(bool opt, string &type)

{

    string username,

        contrasena;

    int contador = 0;

    bool resultado = false;

    do

    {

        vector<string> options = {"USERNAME", "CONTRASENA"};

        vector<string> inputs;

        menuDatos(options, inputs, 1, 1, "##INICIO DE SESIÓN## - ('Salir' para salir) ");

        if (aMinuscula(inputs[0]) == "salir" || aMinuscula(inputs[1]) == "salir")

            break;

        username = inputs[0];

        contrasena = inputs[1];

        resultado = userController.validarSesion(username, contrasena);

        if (resultado)

        {

            system("cls");

            dibujarCuadro();

            centrarTexto("Iniciando Sesión", 0, true, true, -3);

            for (int i = 0; i < 3; i++)

            {

                cout << ".";

                cout.flush();

                sleep(1);

            }

        }

        else

        {

            menuError({"Username y/o contraseña incorrectos."}, 0, "0f", "", true);

            contador++;

        }

    } while (!resultado && contador < 3);

    if (contador >= 3)

    {

        menuError({"Límite de intentos alcanzado, volviendo al menú principal..."}, 0, "0f", "", true);

        return false;

    }

    if (resultado && contador < 3)

    {

        bool isAdm = false, isSupervisor = false;

        Usuario tempUser;

        tempUser = userController.getUsuario(username, contrasena);

        if (tempUser.getTipoUsuario() == "Administrador")

            isAdm = isSupervisor = true;

        else if (tempUser.getTipoUsuario() == "Supervisor")

        {

            isAdm = false;

            isSupervisor = true;

        }

        else

            isAdm = isSupervisor = false;

        menuListado({"Sesion iniciada", "Bienvenido " + username});

        if (opt)

        {

            progController.openSesion(userController.getUsuario(username, contrasena).getCodUsuario(), isSupervisor, isAdm);

            menuMain();

            return true;

        }

        else

        {

            if (isAdm)

                type = "Administrador";

            else if (isSupervisor)

                type = "Supervisor";

            return true;

        }

    }

    return false;

}

void doCerrarSesion()

{

    progController.closeSesion();

    menuLogIn();

}

void doCompra()

{

    int opt;

    vector<string> inputs;

    vector<string> listado;

    long long int proveedorRUC;

    bool usuarioRegistrado;

    int codProducto;

    int cantProducto;

    double montoProducto;

    double montoTotal;

    string comprobante;

    string fecha;

    string observacion = "-";

    int cantComprobantes = kardexController.getCantidadComprobante(false);

    vector<CompraD> carrito;

    vector<kardex> kardexCarrito;

    Compra tempCompra;

    CompraD tempCompraD;

    kardex tempKardex;

    menuListado({"Inicializando Compra..."}, 0, "Compra #" + to\_string(compraController.size() + 1), false);

    Sleep(1000);

    cout << "here" << endl;

    // FECHA

    if (menuConfirmar("Desea usar la fecha de hoy para registrar la compra"))

        fecha = currentDateTime().substr(0, currentDateTime().find(' '));

    else

    {

        do

        {

            inputs.clear();

            menuDatos({"Fecha"}, inputs, 0, 0, "'Salir' para cancelar");

            if (aMinuscula(inputs[0]) == "salir")

                return;

        } while (!kardexController.validarFormatoFecha(inputs[0]) || aMinuscula(inputs[0]) == "salir");

        fecha = inputs[0];

    }

    inputs.clear();

    // RUC

    string nombreProveedor;

    vector<string> listaProveedores;

    for (int i = 0; i < proveedorController.size(); i++)

        listaProveedores.push\_back(proveedorController.get(i).getNombre());

    do

    {

        opt = menu("\_Seleccion de proveedor\_", {"Por Nombre", "Por RUC"});

        switch (opt)

        {

        case 1:

            nombreProveedor = menuBusqueda(listaProveedores, 0, "\_Proveedor\_");

            if (nombreProveedor == "salir")

                opt = -1;

            else

            {

                for (int i = 0; i < proveedorController.size(); i++)

                    if (proveedorController.get(i).getNombre() == nombreProveedor)

                    {

                        proveedorRUC = proveedorController.get(i).getDocumento();

                        break;

                    }

            }

            break;

        case 2:

            do

            {

                usuarioRegistrado = true;

                do

                {

                    inputs.clear();

                    menuDatos({"RUC"}, inputs, 0, 0, "Datos del Proveedor Compra #" + to\_string(compraController.size() + 1)); // Proveedores

                    if (!esNumero(inputs[0]))

                        menuError({"Introduzca un valor numerico"});

                    if (inputs[0].size() != 11)

                        menuError({"El RUC tiene 11 Dígitos"});

                } while (!esNumero(inputs[0]) || inputs[0].size() != 11 || aMinuscula(inputs[0]) == "salir");

                if (aMinuscula(inputs[0]) == "salir")

                    break;

                proveedorRUC = stoll(inputs[0]);

                inputs.clear();

                if (proveedorController.getPosRUC(proveedorRUC) == -1)

                {

                    menuError({"RUC NO REGISTRADO"});

                    if (menuConfirmar("Desea registrar un nuevo Proveedor"))

                    {

                        if (doRegistrarTercero(false, proveedorRUC) == -1)

                            return;

                        usuarioRegistrado = false;

                        menuListado({"Vuelve a buscar el nombre del usuario"});

                    }

                    else

                        return;

                }

            } while (!usuarioRegistrado);

            break;

        default:

            break;

        }

    } while (opt != 1 && opt != 2 && opt != 0);

    if (opt == 0)

        return;

    // COMPROBANTE COMPRA PRODUCTO

    do

    {

        ;

        inputs.clear();

        menuDatos({"Comprobante de Compra"}, inputs, 0, 0, "Datos del Comprobante Compra #" + to\_string(compraController.size() + 1)); // Producto

        if (aMinuscula(inputs[0]) == "salir")

            return;

    } while (!menuConfirmar("Confirma que el comprobante de compra sea " + inputs[0]));

    comprobante = inputs[0];

    inputs.clear();

    string nombreProducto;

    vector<string> listaProductos;

    for (int i = 0; i < productoController.size(); i++)

        listaProductos.push\_back(productoController.get(i).getNombre());

    do

    {

        opt = menu("\_Carrito de Compra\_ #" + to\_string(compraController.size() + 1), {"Agregar producto", "Eliminar Producto", "Mirar Carrito de Compra", "Finalizar Compra"});

        switch (opt)

        {

        case 1:

            // CODIGO PRODUCTO

            nombreProducto = menuBusqueda(listaProductos, 0, "\_Busqueda Producto\_");

            if (nombreProducto == "salir")

                break;

            for (int i = 0; i < productoController.size(); i++)

                if (productoController.get(i).getNombre() == nombreProducto)

                    codProducto = productoController.get(i).getCodProducto();

            // CANTIDAD PRODUCTO

            do

            {

                inputs.clear();

                menuDatos({"Cantidad"}, inputs, 0, 0, "Producto Compra #" + to\_string(compraController.size() + 1)); // Producto

                if (!esNumero(inputs[0]))

                    menuError({"Introduzca un valor numerico"});

                if (inputs[0].size() < 0)

                    menuError({"Cantidad Invalida"});

            } while (!esNumero(inputs[0]) || inputs[0].size() < 0 || aMinuscula(inputs[0]) == "salir");

            if (aMinuscula(inputs[0]) == "salir")

                return;

            cantProducto = stoi(inputs[0]);

            inputs.clear();

            // PRECIO COMPRA PRODUCTO

            do

            {

                inputs.clear();

                menuDatos({"Costo Unitario"}, inputs, 0, 0, "Producto Compra #" + to\_string(compraController.size() + 1)); // Producto

                if (!esNumero(inputs[0]))

                    menuError({"Introduzca un valor numerico"});

                if (inputs[0].size() < 0)

                    menuError({"Cantidad Invalida"});

            } while (!esNumero(inputs[0]) || inputs[0].size() < 0 || aMinuscula(inputs[0]) == "salir");

            if (aMinuscula(inputs[0]) == "salir")

                return;

            montoProducto = stoi(inputs[0]);

            inputs.clear();

            tempCompraD.setCodCompra(compraController.size());

            tempCompraD.setCodProducto(codProducto);

            tempCompraD.setCantidad(cantProducto);

            tempCompraD.setPrecio(montoProducto);

            tempCompraD.setMonto(montoProducto \* cantProducto);

            carrito.push\_back(tempCompraD);

            break;

        case 2:

            do

            {

                listado.clear();

                for (CompraD x : carrito)

                    listado.push\_back(productoController.get(x.getCodProducto()).getNombre());

                opt = menu("Suprimir Productos Compra #" + to\_string(compraController.size() + 1), listado);

                if (opt != 0 && opt != -1)

                {

                    carrito.erase(carrito.begin() + opt - 1);

                    listado.erase(listado.begin() + opt - 1);

                }

            } while (opt != 0);

            opt = 2;

            break;

        case 3:

            listado.clear();

            for (CompraD x : carrito)

                listado.push\_back(productoController.get(x.getCodProducto()).getNombre());

            menuListado(listado, 0, "Compra #" + to\_string(compraController.size() + 1));

            break;

        case 4:

            listado.clear();

            for (CompraD x : carrito)

                listado.push\_back(productoController.get(x.getCodProducto()).getNombre());

            menuListado(listado, 0, "Compra #" + to\_string(compraController.size() + 1));

            if (menuConfirmar("Desea agregar una observacion a esta compra"))

            {

                ;

                menuDatos({""}, inputs, 0, 0, "Observacion Compra #" + to\_string(compraController.size() + 1));

                observacion = inputs[0];

            }

            montoTotal = 0;

            for (CompraD x : carrito)

            {

                tempKardex.setCantidad(x.getCantidad());

                tempKardex.setMontoUnitario(x.getPrecio());

                tempKardex.setCodProducto(x.getCodProducto());

                tempKardex.setCodProceso(createCode("CP", compraController.size(), 5));

                tempKardex.setComprobante(comprobante);

                tempKardex.setFechaDeEmision(fecha);

                tempKardex.setIsSalida(false);

                tempKardex.setMontoUnitario(x.getPrecio());

                tempKardex.setMotivo("~~Compra " + to\_string(cantComprobantes) + "~~");

                tempKardex.setObservacion(observacion);

                kardexController.add(tempKardex);

                compraDController.add(x);

                kardexController.saveFile();

                compraDController.saveFile();

                montoTotal += (x.getPrecio() \* x.getCantidad());

            }

            tempCompra.setCodCompra(compraController.size());

            tempCompra.setCodProveedor(proveedorController.getPosRUC(proveedorRUC));

            tempCompra.setCodUsuario(progController.getSesionKey());

            tempCompra.setFecha(fecha);

            tempCompra.setMonto(montoTotal);

            tempCompra.setEstado(true);

            compraController.add(tempCompra);

            compraController.saveFile();

            opt = 0;

            break;

        case 0:

            if (!menuConfirmar("Seguro que desea cancelar la compra"))

                opt = -1;

            break;

        default:

            break;

        }

    } while (opt != 0);

}

void doVenta()

{

    int opt;

    vector<string> inputs;

    vector<string> listado;

    vector<string> options;

    string tipoDocumento;

    bool usuarioRegistrado;

    long long int clienteDNI;

    int codProducto;

    int cantProducto;

    double montoProducto;

    double montoTotal;

    string comprobante;

    string fecha;

    string observacion = "-";

    int cantComprobantes = kardexController.getCantidadComprobante(true);

    vector<VentaD> carrito;

    vector<kardex> kardexCarrito;

    Venta tempVenta;

    VentaD tempVentaD;

    kardex tempKardex;

    menuListado({"Inicializando Venta..."}, 0, "Venta #" + to\_string(ventaController.size() + 1), false);

    Sleep(1000);

    cout << "here" << endl;

    // FECHA

    if (menuConfirmar("Desea usar la fecha de hoy para registrar la Venta"))

        fecha = currentDateTime().substr(0, currentDateTime().find(' '));

    else

    {

        do

        {

            inputs.clear();

            menuDatos({"Fecha"}, inputs, 0, 0, "'Salir' para cancelar");

            if (aMinuscula(inputs[0]) == "salir")

                return;

        } while (!kardexController.validarFormatoFecha(inputs[0]) || aMinuscula(inputs[0]) == "salir");

        fecha = inputs[0];

    }

    inputs.clear();

    // DNI

    string nombreCliente;

    vector<string> listaClientes;

    for (int i = 0; i < clienteController.size(); i++)

        listaClientes.push\_back(clienteController.get(i).getNombre());

    do

    {

        opt = menu("\_Seleccion de cliente\_", {"Por Nombre", "Por Documento"});

        switch (opt)

        {

        case 1:

            nombreCliente = menuBusqueda(listaClientes, 0, "\_Cliente\_");

            if (nombreCliente == "salir")

                opt = -1;

            else

            {

                for (int i = 0; i < clienteController.size(); i++)

                    if (clienteController.get(i).getNombre() == nombreCliente)

                    {

                        cout << clienteController.get(i).getDocumento() << endl;

                        clienteDNI = clienteController.get(i).getDocumento();

                        tipoDocumento = clienteController.get(i).getTipoDocumento();

                        break;

                    }

            }

            break;

        case 2:

            do

            {

                usuarioRegistrado = true;

                options = {"DNI", "CARNET EXTRANJERIA (CE)", "PASAPORTE", "RUC"}; // Carritos de compra, etc

                opt = menu("REGISTRO - TIPO DE USUARIO", options);

                switch (opt)

                {

                case 1:

                    tipoDocumento = "DNI";

                    break;

                case 2:

                    tipoDocumento = "CE";

                    break;

                case 3:

                    tipoDocumento = "PASAPORTE";

                    break;

                case 4:

                    tipoDocumento = "RUC";

                    break;

                case 0:

                    tipoDocumento = "Cancelar";

                    break;

                }

                if (tipoDocumento == "Cancelar")

                    return;

                do

                {

                    inputs.clear();

                    menuDatos({"Documento"}, inputs, 0, 0, "Datos del Cliente Venta #" + to\_string(ventaController.size() + 1)); // Clientes

                    if (!esNumero(inputs[0]))

                        menuError({"Introduzca un valor numerico"});

                    // if (inputs[0].size() != 8)

                    //     menuError({"El DNI tiene 8 Dígitos"});

                } while (!esNumero(inputs[0]) /\*|| inputs[0].size() != 8\*/ || aMinuscula(inputs[0]) == "salir");

                if (aMinuscula(inputs[0]) == "salir")

                    break;

                clienteDNI = stoll(inputs[0]);

                inputs.clear();

                if (clienteController.getPosDoc(clienteDNI, tipoDocumento) == -1)

                {

                    menuError({aMayuscula(tipoDocumento) + " NO REGISTRADO"});

                    if (menuConfirmar("Desea registrar un nuevo Cliente"))

                    {

                        if (doRegistrarTercero(true, clienteDNI) == -1)

                            return;

                        usuarioRegistrado = false;

                        menuListado({"Vuelve a buscar el nombre del usuario"});

                    }

                    else

                        return;

                }

            } while (!usuarioRegistrado);

            break;

        default:

            break;

        }

    } while (opt != 1 && opt != 2 && opt != 0);

    if (opt == 0)

        return;

    // COMPROBANTE VENTA PRODUCTO

    inputs.clear();

    if (tipoDocumento == "RUC")

    {

        comprobante = createCode("FV", kardexController.getCantidadComprobanteVenta(false));

    }

    else

    {

        comprobante = createCode("BV", kardexController.getCantidadComprobanteVenta(true));

    }

    string nombreProducto;

    vector<string> listaProductos;

    for (int i = 0; i < productoController.size(); i++)

        listaProductos.push\_back(productoController.get(i).getNombre());

    do

    {

        opt = menu("\_Carrito de " + clienteController.get(clienteController.getPosDoc(clienteDNI, tipoDocumento)).getNombre() + " de Venta\_ #" + to\_string(ventaController.size() + 1), {"Agregar producto", "Eliminar Producto", "Mirar Carrito de Compra", "Finalizar Compra"});

        switch (opt)

        {

        case 1:

            // CODIGO PRODUCTO

            nombreProducto = menuBusqueda(listaProductos, 0, "\_Busqueda Producto\_");

            if (nombreProducto == "salir")

                break;

            for (int i = 0; i < productoController.size(); i++)

                if (productoController.get(i).getNombre() == nombreProducto)

                    codProducto = productoController.get(i).getCodProducto();

            // CANTIDAD PRODUCTO

            do

            {

                inputs.clear();

                menuDatos({"Cantidad"}, inputs, 0, 0, "Producto Venta #" + to\_string(ventaController.size() + 1)); // Producto

                if (!esNumero(inputs[0]))

                    menuError({"Introduzca un valor numerico"});

                if (inputs[0].size() < 0)

                    menuError({"Cantidad Invalida"});

            } while (!esNumero(inputs[0]) || inputs[0].size() < 0 || aMinuscula(inputs[0]) == "salir");

            if (aMinuscula(inputs[0]) == "salir")

                return;

            cantProducto = stoi(inputs[0]);

            inputs.clear();

            // // PRECIO COMPRA PRODUCTO

            // do

            // {

            //     inputs.clear();

            //     menuDatos({"Costo Unitario"}, inputs, 0, 0, "Producto Compra #" + to\_string(ventaController.size() + 1)); // Producto

            //     if (!esNumero(inputs[0]))

            //         menuError({"Introduzca un valor numerico"});

            //     if (inputs[0].size() < 0)

            //         menuError({"Cantidad Invalida"});

            // } while (!esNumero(inputs[0]) || inputs[0].size() < 0 || aMinuscula(inputs[0]) == "salir");

            // if (aMinuscula(inputs[0]) == "salir")

            //     return;

            // montoProducto = stoi(inputs[0]);

            // inputs.clear();

            tempVentaD.setCodVenta(ventaController.size());

            tempVentaD.setCodProducto(codProducto);

            tempVentaD.setCantidad(cantProducto);

            tempVentaD.setPrecio(productoController.get(codProducto).getPrecioUnitario());

            tempVentaD.setMonto(productoController.get(codProducto).getPrecioUnitario() \* cantProducto);

            carrito.push\_back(tempVentaD);

            break;

        case 2:

            do

            {

                listado.clear();

                for (VentaD x : carrito)

                    listado.push\_back(productoController.get(x.getCodProducto()).getNombre());

                opt = menu("Suprimir Productos Venta #" + to\_string(ventaController.size() + 1), listado);

                if (opt != 0 && opt != -1)

                {

                    carrito.erase(carrito.begin() + opt - 1);

                    listado.erase(listado.begin() + opt - 1);

                }

            } while (opt != 0);

            opt = 2;

            break;

        case 3:

            listado.clear();

            for (VentaD x : carrito)

                listado.push\_back(productoController.get(x.getCodProducto()).getNombre());

            menuListado(listado, 0, "Venta #" + to\_string(ventaController.size() + 1));

            break;

        case 4:

            listado.clear();

            for (VentaD x : carrito)

                listado.push\_back(productoController.get(x.getCodProducto()).getNombre());

            menuListado(listado, 0, "Venta #" + to\_string(ventaController.size() + 1));

            if (menuConfirmar("Desea agregar una observacion a esta Venta"))

            {

                menuDatos({""}, inputs, 0, 0, "Observacion Venta #" + to\_string(ventaController.size() + 1));

                observacion = inputs[0];

            }

            for (VentaD x : carrito)

            {

                tempKardex.setCantidad(x.getCantidad());

                tempKardex.setMontoUnitario(x.getPrecio());

                tempKardex.setCodProducto(x.getCodProducto());

                tempKardex.setCodProceso(createCode("VT", ventaController.size(), 5));

                tempKardex.setComprobante(comprobante);

                tempKardex.setFechaDeEmision(fecha);

                tempKardex.setIsSalida(true);

                tempKardex.setMontoUnitario(x.getPrecio());

                tempKardex.setMotivo("~~Venta " + to\_string(cantComprobantes) + "~~");

                tempKardex.setObservacion(observacion);

                kardexController.add(tempKardex);

                ventaDController.add(x);

                kardexController.saveFile();

                ventaDController.saveFile();

                montoTotal += (x.getPrecio() \* x.getCantidad());

            }

            tempVenta.setCodVenta(ventaController.size());

            tempVenta.setCodCliente(clienteController.getPosDoc(clienteDNI, tipoDocumento));

            tempVenta.setCodUsuario(progController.getSesionKey());

            tempVenta.setFecha(fecha);

            tempVenta.setMonto(montoTotal);

            tempVenta.setEstado(true);

            ventaController.add(tempVenta);

            ventaController.saveFile();

            opt = 0;

            break;

        case 0:

            if (!menuConfirmar("Seguro que desea cancelar la Venta"))

                opt = -1;

            break;

        default:

            break;

        }

    } while (opt != 0);

}

void doMovimiento()

{

    int opt;

    vector<string> inputs;

    vector<string> listado;

    long long int clienteDNI;

    int codProducto;

    int cantProducto;

    double montoProducto;

    string comprobante;

    string codigoMovimiento;

    string fecha;

    string motivo;

    string observacion = "-";

    int cantComprobantes = kardexController.getCantidadComprobante(true);

    vector<CompraD> carrito;

    vector<kardex> kardexCarrito;

    CompraD tempCompraD;

    kardex tempKardex;

    // FECHA

    if (menuConfirmar("Desea usar la fecha de hoy para registrar el Movimiento"))

    {

        cin.ignore();

        fecha = currentDateTime().substr(0, currentDateTime().find(' '));

    }

    else

    {

        cin.ignore();

        do

        {

            cout << kardexController.validarFormatoFecha(inputs[0]);

            system("pause");

            inputs.clear();

            menuDatos({"Fecha"}, inputs, 0, 0, "'Salir' para cancelar");

        } while (!kardexController.validarFormatoFecha(inputs[0]) || aMinuscula(inputs[0]) == "salir");

        if (aMinuscula(inputs[0]) == "salir")

            return;

        fecha = inputs[0];

    }

    inputs.clear();

    // DOCUMENTOS

    do

    {

        inputs.clear();

        menuDatos({"Documento"}, inputs, 0, 0, "Datos del Documento Movimiento #" + to\_string(kardexController.getVector().size() + 1)); // Clientes

        if (!esNumero(inputs[0]))

            menuError({"Introduzca un valor numerico"});

    } while (!esNumero(inputs[0]) || aMinuscula(inputs[0]) == "salir");

    if (aMinuscula(inputs[0]) == "salir")

        return;

    clienteDNI = stoll(inputs[0]);

    inputs.clear();

    // COMPROBANTE VENTA PRODUCTO

    cin.ignore();

    inputs.clear();

    opt = menu("Comprobante Movimiento #" + to\_string(kardexController.getVector().size() + 1), {"Manualmente", "Boleta de Venta Automatica", "Factura de Venta Automatica", "Ninguno"});

    switch (opt)

    {

    case 1:

        do

        {

            cin.ignore();

            inputs.clear();

            menuDatos({"Comprobante de Compra"}, inputs, 0, 0, "Datos del Comprobante Movimiento #" + to\_string(kardexController.getVector().size() + 1)); // Producto

            if (aMinuscula(inputs[0]) == "salir")

                return;

        } while (!menuConfirmar("Confirma que el comprobante de compra sea " + inputs[0]));

        comprobante = inputs[0];

    case 2:

        comprobante = createCode("BV", kardexController.getCantidadComprobanteVenta(true));

        break;

    case 3:

        comprobante = createCode("FV", kardexController.getCantidadComprobanteVenta(false));

        break;

    case 4:

        comprobante = "-";

    default:

        return;

        break;

    }

    // CODIGO MOVIMIENTO

    cin.ignore();

    inputs.clear();

    opt = menu("Codigo Movimiento #" + to\_string(kardexController.getVector().size() + 1), {"Manualmente", "Automatico", "Ninguno"});

    switch (opt)

    {

    case 1:

        do

        {

            cin.ignore();

            inputs.clear();

            menuDatos({"Codigo de Movimiento"}, inputs, 0, 0, "Codigo del Movimiento #" + to\_string(kardexController.getVector().size() + 1)); // Producto

            if (aMinuscula(inputs[0]) == "salir")

                return;

        } while (!menuConfirmar("Confirma que el comprobante de compra sea " + inputs[0]));

    case 2:

        codigoMovimiento = createCode("MT", kardexController.getCantidadComprobante(false, true), 5);

        break;

    case 3:

        codigoMovimiento = "-";

    default:

        return;

        break;

    }

    //  PRODUCTO

    do

    {

        inputs.clear();

        menuDatos({"Producto"}, inputs, 0, 0, "Producto Movimiento #" + to\_string(kardexController.getVector().size() + 1)); // Producto

        if (!esNumero(inputs[0]))

            menuError({"Introduzca un valor numerico"});

        if (stoi(inputs[0]) >= productoController.getNewCodProducto())

            menuError({"Producto no registrado"});

    } while (!esNumero(inputs[0]) || stoi(inputs[0]) >= productoController.getNewCodProducto() || aMinuscula(inputs[0]) == "salir");

    if (aMinuscula(inputs[0]) == "salir")

        return;

    codProducto = stoi(inputs[0]);

    inputs.clear();

    bool isSalida = menuConfirmar("El movimiento es una reduccion en el inventario");

    // CANTIDAD PRODUCTO

    do

    {

        inputs.clear();

        cin.ignore();

        menuDatos({"Cantidad"}, inputs, 0, 0, "Producto Movimiento #" + to\_string(kardexController.getVector().size() + 1)); // Producto

        if (!esNumero(inputs[0]))

            menuError({"Introduzca un valor numerico"});

        if (inputs[0].size() < 0)

            menuError({"Cantidad Invalida"});

    } while (!esNumero(inputs[0]) || inputs[0].size() < 0 || aMinuscula(inputs[0]) == "salir");

    if (aMinuscula(inputs[0]) == "salir")

        return;

    cantProducto = stoi(inputs[0]);

    inputs.clear();

    // PRECIO PRODUCTO

    cin.ignore();

    inputs.clear();

    opt = menu("Agregar precio al Producto, Movimiento #" + to\_string(kardexController.getVector().size() + 1), {"Manualmente", "Automatico", "Ninguno"});

    switch (opt)

    {

    case 1:

        do

        {

            inputs.clear();

            menuDatos({"Costo Unitario"}, inputs, 0, 0, "Producto Movimiento #" + to\_string(kardexController.getVector().size() + 1)); // Producto

            if (!esNumero(inputs[0]))

                menuError({"Introduzca un valor numerico"});

            if (inputs[0].size() < 0)

                menuError({"Cantidad Invalida"});

        } while (!esNumero(inputs[0]) || inputs[0].size() < 0 || aMinuscula(inputs[0]) == "salir");

        if (aMinuscula(inputs[0]) == "salir")

            return;

        montoProducto = stoi(inputs[0]);

    case 2:

        montoProducto = productoController.get(codProducto).getPrecioUnitario();

        break;

    case 3:

        montoProducto = 0;

        break;

    default:

        return;

        break;

    }

    inputs.clear();

    tempCompraD.setCodCompra(compraController.size());

    tempCompraD.setCodProducto(codProducto);

    tempCompraD.setCantidad(cantProducto);

    tempCompraD.setPrecio(montoProducto);

    carrito.push\_back(tempCompraD);

    ///////////////////////////////////////////////////////////

    cin.ignore();

    menuDatos({""}, inputs, 0, 0, "Motivo Movimiento #" + to\_string(kardexController.getVector().size() + 1));

    motivo = inputs[0];

    if (menuConfirmar("Desea agregar una observacion a este Movimiento"))

    {

        cin.ignore();

        menuDatos({""}, inputs, 0, 0, "Observacion Movimiento #" + to\_string(kardexController.getVector().size() + 1));

        observacion = inputs[0];

    }

    for (CompraD x : carrito)

    {

        tempKardex.setCantidad(x.getCantidad());

        tempKardex.setMontoUnitario(x.getPrecio());

        tempKardex.setCodProducto(x.getCodProducto());

        tempKardex.setCodProceso(codigoMovimiento);

        tempKardex.setComprobante(comprobante);

        tempKardex.setFechaDeEmision(fecha);

        tempKardex.setIsSalida(isSalida);

        tempKardex.setMontoUnitario(x.getPrecio());

        tempKardex.setMotivo(motivo);

        tempKardex.setObservacion(observacion);

        kardexController.add(tempKardex);

        kardexController.saveFile();

    }

}

void doRegistrarRetiroCaja()

{

    double monto;

    string observacion;

    vector<string> inputs;

    if (!progController.getActiveSesion() || !progController.getIsSupervisor())

    {

        menuError({"~-NECESITAS TENER UNA SESIÓN DE SUPERVISOR O ADMINISTRADOR INICIADA-~"});

        return;

    }

    do

    {

        inputs.clear();

        menuDatos({"Monto de Retiro"}, inputs, 0, 0, "Retiro de Efectivo"); // Producto

        if (!esNumero(inputs[0]))

            menuError({"Introduzca un valor numerico"});

    } while (!esNumero(inputs[0]) || aMinuscula(inputs[0]) == "salir");

    if (aMinuscula(inputs[0]) == "salir")

        return;

    monto = stoi(inputs[0]);

    inputs.clear();

    if (menuConfirmar("Desea agregar una observacion a este Movimiento"))

    {

        cin.ignore();

        menuDatos({""}, inputs, 0, 0, "Observacion - Retiro de Efectivo");

        observacion = inputs[0];

    }

    inputs.clear();

    kardex obj(currentDateTime().substr(0, currentDateTime().find(' ')), 1, -1, monto, createCode("RT", kardexController.getMovimientosRetiros().size(), 5),

               createCode("RT", kardexController.getMovimientosRetiros().size(), 5), true,

               "RETIRO DE EFECTIVO ( S/." + to\_string(monto) + " ) POR " + userController.get(progController.getSesionKey()).getNombre() +

                   " " + userController.get(progController.getSesionKey()).getApellidos(),

               observacion);

    kardexController.add(obj);

    kardexController.saveFile();

}

void askEstadoCaja()

{

    menuListado({"S/" + to\_string(kardexController.getEstadoCaja())}, 0, "\_SALDO ACTUAL\_", true);

}

void doConsultarRegistro(int cod)

{

    int opt;

    int codUsuario = cod;

    vector<Venta> registro;

    vector<string> listaRegistro;

    for (int i = 0; i < ventaController.size(); i++)

    {

        Venta vTemp = ventaController.get(i);

        if (vTemp.getCodUsuario() == codUsuario)

            registro.push\_back(vTemp);

    }

    if (registro.size() == 0)

    {

        menuError({"Aun no se hicieron ventas"});

    }

    else

    {

        for (Venta x : registro)

            listaRegistro.push\_back(to\_string(x.getCodVenta()) + "\tS/" + to\_string(x.getMonto()));

        do

        {

            vector<string> listaVenta;

            opt = menu("\_REGISTRO DE VENTAS\_", listaRegistro);

            switch (opt)

            {

            case -1:

                break;

            case 0:

                break;

            default:

                Venta vTemp;

                Cliente cTemp;

                vector<VentaD> registroVentaD;

                vTemp = ventaController.get(opt - 1);

                cTemp = clienteController.get(vTemp.getCodCliente() - 1);

                listaVenta.push\_back("Nombre del Cliente: " + cTemp.getNombre());

                listaVenta.push\_back("DNI del Cliente: " + cTemp.getDocumento());

                listaVenta.push\_back("Productos:");

                for (int i = 0; i < ventaDController.size(); i++)

                {

                    VentaD vDTemp = ventaDController.get(i);

                    if (vDTemp.getCodVenta() == opt)

                    {

                        Producto pTemp;

                        pTemp = productoController.get(vDTemp.getCodProducto() - 1);

                        listaVenta.push\_back("\t" + pTemp.getNombre() + "\tS/" + to\_string(vDTemp.getPrecio()) + "\t" + to\_string(vDTemp.getCantidad()) + "u\tS/" + to\_string(vDTemp.getMonto()));

                    }

                }

                listaVenta.push\_back("Monto total: S/" + to\_string(vTemp.getMonto()));

                menuListado(listaVenta);

                break;

            }

        } while (opt != 0);

    }

}

void doBuscarRegistro()

{

    int opt;

    vector<string> usuarios;

    for (int i = 0; i < userController.size(); i++)

    {

        Usuario uTemp = userController.get(i);

        usuarios.push\_back(uTemp.getUsername() + "\t" + uTemp.getNombre() + " " + uTemp.getApellidos());

    }

    do

    {

        opt = menu("\_USUARIOS\_", usuarios);

        switch (opt)

        {

        case -1:

            break;

        case 0:

            break;

        default:

            doConsultarRegistro(opt - 1);

            break;

        }

    } while (opt != 0);

}

void askInventario()

{

    int cod;

    string producto;

    vector<string> listaProductos;

    for (int i = 0; i < productoController.size(); i++)

        listaProductos.push\_back(productoController.get(i).getNombre());

    if (listaProductos.size() != 0)

    {

        do

        {

            producto = menuBusqueda(listaProductos, 0, "\_Buscar productos\_");

            if (producto != "salir")

            {

                for (int i = 0; i < productoController.size(); i++)

                    if (productoController.get(i).getNombre() == producto)

                        cod = productoController.get(i).getCodProducto();

                int j = 1;

                Producto temp;

                temp = productoController.get(cod);

                vector<string> listado{"Codigo de producto: " + to\_string(temp.getCodProducto()),

                                       "Marca: " + marcaController.get(temp.getCodMarca()).getNombreMarca(),

                                       "Nombre: " + temp.getNombre(),

                                       "Precio unitario: S/" + to\_string(temp.getPrecioUnitario()),

                                       "Unidades disponibles: " + to\_string(temp.getStock())};

                for (Componente x : temp.getVectorComponentes())

                {

                    listado.push\_back("Componente[" + to\_string(j) + "]: " + x.getNombre() + "\t" + x.getCantidad());

                    j++;

                }

                menuListado(listado, 0, aMayuscula(temp.getNombre()), true);

            }

        } while (producto != "salir");

    }

    else

    {

        menuError({"No hay productos registrados, volviendo al menu inventario"});

    }

}

void changeDataInventario()

{

    int opt;

    int codigo;

    vector<string> inputs;

    string temporal;

    bool salida;

    vector<Componente> componentes;

    if (!progController.getActiveSesion() || !progController.getIsSupervisor())

    {

        menuError({"Necesitas iniciar sesión como Supervisor"});

        if (!doIniciarSesion(false, temporal))

            return;

        else if (temporal != "Supervisor" && temporal != "Admin")

            return;

    }

    string producto;

    vector<string> listaProductos;

    for (int i = 0; i < productoController.size(); i++)

        listaProductos.push\_back(productoController.get(i).getNombre());

    do

    {

        opt = menu("\_Busqueda de Producto\_", {"Por codigo", "Por nombre"});

        switch (opt)

        {

        case 1:

            do

            {

                salida = true;

                do

                {

                    menuDatos({"Codigo Producto"}, inputs, 0, 0, "Cambiar Datos Producto -\"Salir\" para Salir");

                    if (aMinuscula(inputs[0]) == "salir")

                        break;

                    if (!esNumero(inputs[0]))

                    {

                        menuError({"Introduce un valor numérico"});

                        inputs[0] = "-1";

                    }

                    else if (stoi(inputs[0]) < 0 || stoi(inputs[0]) >= productoController.getNewCodProducto())

                        menuError({"Codigo Fuera de Rango"});

                } while ((stoi(inputs[0]) < 0 || stoi(inputs[0]) >= productoController.getNewCodProducto()) && aMinuscula(inputs[0]) != "salir");

                if (aMinuscula(inputs[0]) == "salir")

                    return;

                if (!menuConfirmar("El producto consultado es " + productoController.get(stoi(inputs[0])).getNombre()))

                {

                    if (!menuConfirmar("Desea continuar con el proceso"))

                        return;

                    else

                    {

                        salida = false;

                    }

                }

            } while (!salida);

            codigo = stoi(inputs[0]);

            break;

        case 2:

            producto = menuBusqueda(listaProductos, 0, "\_Buscar productos\_");

            if (producto != "salir")

            {

                for (int i = 0; i < productoController.size(); i++)

                    if (productoController.get(i).getNombre() == producto)

                    {

                        codigo = productoController.get(i).getCodProducto();

                        break;

                    }

            }

            break;

        default:

            break;

        }

    } while (opt != 1 && opt != 2 && opt != 0);

    if (opt == 0)

        return;

    Producto prodTemporal = productoController.get(codigo);

    if (productoController.getNewCodProducto() != 0)

    {

        do

        {

            vector<string> options = {"Editar Nombre", "Modificar Marca", "Editar Precios", "Editar Componentes"};

            opt = menu("Selecciona que deseas modificar", options);

            inputs.clear();

            componentes.clear();

            switch (opt)

            {

            case 1:

                menuDatos({"Nuevo nombre de producto"}, inputs);

                temporal = inputs[0];

                if (menuConfirmar("Desea que \"" + temporal + "\" sea el nuevo nombre del producto"))

                    prodTemporal.setNombre(temporal);

                break;

            case 2:

                do

                {

                    menuDatos({"Nuevo código de marca"}, inputs);

                    temporal = inputs[0];

                    if (!esNumero(temporal))

                    {

                        menuError({"Introduce un valor numérico"});

                        temporal = "-1";

                    }

                    else if (stoi(temporal) < 0 || stoi(temporal) >= marcaController.getNewCodMarca())

                        menuError({"Codigo Fuera de Rango"});

                } while (stoi(temporal) < 0 || stoi(temporal) >= marcaController.getNewCodMarca() || temporal == "");

                if (menuConfirmar("Desea que\"" + marcaController.get(stoi(temporal)).getNombreMarca() + "\"sea la nueva marca"))

                    prodTemporal.setCodMarca(stoi(temporal));

                break;

            case 3:

                do

                {

                    menuDatos({"Nuevo precio Unitario"}, inputs);

                    temporal = inputs[0];

                    if (!esNumero(temporal))

                    {

                        menuError({"Introduce un valor numérico"});

                        temporal = "-1";

                    }

                } while (stod(temporal) < 0 || temporal == "");

                if (menuConfirmar("Desea que \"" + temporal + "\" sea el nuevo Precio"))

                    prodTemporal.setPrecioUnitario(stod(temporal));

                break;

            case 4:

                do

                {

                    int i = 0;

                    menuDatos({"Nombre del componente", "Cantidad del componente"}, inputs, 0, 0, "\_COMPONENTE [" + to\_string(i + 1) + "]\_");

                    Componente temp(inputs[0], inputs[1]);

                    componentes.push\_back(temp);

                    i++;

                } while (menuConfirmar("Desea agregar un nuevo componente"));

                if (menuConfirmar("Desea modificar los componentes"))

                    prodTemporal.setComponentes(componentes);

                break;

            case 0:

                productoController.modify(prodTemporal, codigo);

                productoController.saveFile();

                break;

            }

        } while (opt != 0);

    }

    else

    {

        menuError({"NO HAY PRODUCTOS REGISTRADOS"});

    }

}

void doAddProducto()

{

    int codMarca;

    int opt;

    int i;

    string temporal;

    vector<Componente> componentes;

    vector<string> inputs;

    vector<string> optionsM;

    if (!progController.getActiveSesion() || !progController.getIsSupervisor())

    {

        menuError({"Necesitas iniciar sesión como Supervisor"});

        if (!doIniciarSesion(false, temporal))

            return;

        else if (temporal != "Supervisor" && temporal != "Admin")

            return;

    }

    for (int i = 0; i < marcaController.getNewCodMarca(); i++)

        optionsM.push\_back(marcaController.get(i).getNombreMarca());

    do

    {

        int codigo = productoController.getNewCodProducto();

        componentes.clear();

        i = 1;

        inputs.clear();

        ;

        menuDatos({"Nombre del Producto"}, inputs, 0, 0, "\_PRODUCTO [" + to\_string(codigo + 1) + "]\_");

        string nombre = inputs[0];

        do

        {

            inputs.clear();

            ;

            menuDatos({"Nombre del componente", "Cantidad del componente"}, inputs, 0, 0, "\_COMPONENTE [" + to\_string(i) + "]\_");

            Componente temp(inputs[0], inputs[1]);

            componentes.push\_back(temp);

            i++;

        } while (menuConfirmar("Desea agregar un nuevo componente"));

        if (marcaController.getNewCodMarca() != 0)

        {

            opt = menu("AGREGAR MARCAS", optionsM);

            if (opt != 0 && opt != -1)

            {

                codMarca = opt - 1;

                ;

                do

                {

                    inputs.clear();

                    menuDatos({"Precio S/"}, inputs);

                    if (!esNumero(inputs[0]))

                        menuError({"Introduce un valor numérico"});

                } while (!esNumero(inputs[0]));

                double precio = stod(inputs[0]);

                Producto producto(productoController.getNewCodProducto(), codMarca, nombre, precio, 0, componentes);

                productoController.add(producto);

            }

        }

    } while (menuConfirmar("Desea agregar un nuevo producto"));

    productoController.saveFile();

}

void doAddMarca()

{

    int codigo;

    vector<string> inputs;

    string temporal;

    if (!progController.getActiveSesion() || !progController.getIsSupervisor())

    {

        menuError({"Necesitas iniciar sesión como Supervisor"});

        if (!doIniciarSesion(false, temporal))

            return;

        else if (temporal != "Supervisor" && temporal != "Admin")

            return;

    }

    do

    {

        codigo = marcaController.getNewCodMarca();

        inputs.clear();

        ;

        menuDatos({"Nombre de la marca"}, inputs, 0, 0, "\_MARCA [" + to\_string(codigo + 1) + "]\_");

        Marca marca(codigo, inputs[0]);

        marcaController.add(marca);

    } while (menuConfirmar("Desea agregar una nueva marca"));

    marcaController.saveFile();

}

long long int doRegistrarTercero(bool isCliente, long long int documento = -1)

{

    string tipoDocumento, strInput, nombreTercero;

    int docSize, opt;

    long long int numeroDocumento;

    vector<string> options, inputs;

    Cliente objCliente;

    Proveedor objProveedor;

    bool duplicado;

    if (documento != -1)

        numeroDocumento == documento;

    // INTRODUCIR NUMERO DOCUMENTO

    strInput = "0";

    do

    {

        do

        {

            if (isCliente)

            {

                options = {"DNI", "CARNET EXTRANJERIA (CE)", "PASAPORTE", "RUC"}; // Carritos de compra, etc

                opt = menu("REGISTRO - TIPO DE USUARIO", options);

                switch (opt)

                {

                case 1:

                    tipoDocumento = "DNI";

                    docSize = 8;

                    break;

                case 2:

                    tipoDocumento = "CE";

                    docSize = 12;

                    break;

                case 3:

                    tipoDocumento = "PASAPORTE";

                    docSize = 12;

                    break;

                case 4:

                    tipoDocumento = "RUC";

                    docSize = 11;

                    break;

                case 0:

                    tipoDocumento = "Cancelar";

                    break;

                }

                if (tipoDocumento == "Cancelar")

                    return -1;

            }

            else

            {

                tipoDocumento = "RUC";

                docSize = 11;

            }

            inputs.clear();

            if (documento < 0)

            {

                menuDatos({"Numero de documento"}, inputs, 0, 0, "\_DOCUMENTO\_ -('Salir' para salir)");

                strInput = inputs[0].substr(0, inputs[0].find(' '));

            }

            else

            {

                strInput = to\_string(documento);

            }

            if (aMinuscula(strInput) == "salir")

                return -1;

            if ((int)strInput.size() != docSize)

            {

                options = {"~-Los documentos tipo " + tipoDocumento + " deben contener " + to\_string(docSize) + " digitos-~", "~VUELVA A INGRESAR SU NUMERO DE DOCUMENTO~"};

                menuError(options, 1);

                documento = -1;

            }

            if (!esNumero(strInput))

                menuError({"~[[INTRODUCE UN VALOR NUMERICO]]~"});

        } while (!esNumero(strInput) || (int)strInput.size() != docSize);

        numeroDocumento = stoll(strInput);

        if (isCliente)

        {

            if (clienteController.getPosDoc(numeroDocumento, tipoDocumento) != -1)

            {

                duplicado = true;

            }

            else

                duplicado = false;

        }

        else

        {

            if (proveedorController.getPosRUC(numeroDocumento) != -1)

                duplicado = true;

            else

                duplicado = false;

        }

        if (duplicado)

        {

            menuError({"Ya Registrado, 'salir' para salir"});

            documento = -1;

        }

    } while (duplicado);

    // NOMBRE

    do

    {

        inputs.clear();

        menuDatos({"Nombre"}, inputs, 0, 0, ""); // Producto

        if (aMinuscula(inputs[0]) == "salir")

            return -1;

    } while (!menuConfirmar("Confirma que el nombre es " + inputs[0]));

    nombreTercero = inputs[0];

    inputs.clear();

    if (isCliente)

    {

        objCliente.setCodCliente(clienteController.size());

        objCliente.setTipoDocumento(tipoDocumento);

        objCliente.setDocumento(numeroDocumento);

        objCliente.setNombre(nombreTercero);

        clienteController.add(objCliente);

        clienteController.saveFile();

    }

    else

    {

        objProveedor.setCodProveedor(proveedorController.size());

        objProveedor.setTipoDocumento(tipoDocumento);

        objProveedor.setDocumento(numeroDocumento);

        objProveedor.setNombre(nombreTercero);

        proveedorController.add(objProveedor);

        proveedorController.saveFile();

    }

    return numeroDocumento;

}

void doActualizarInventario()

{

    for (int i = 0; i < productoController.size(); i++)

    {

        Producto objProducto = productoController.get(i);

        objProducto.setStock(kardexController.getCantidadProducto(i));

        productoController.modify(objProducto, i);

    }

    productoController.saveFile();

}

##### OptionsMenu.h

#ifndef OPTIONMENUS\_CPP

#define OPTIONMENUS\_CPP

#include <iostream>

#include <string>

#include "../model/FuncionesGenerales.h"

#include "Funciones.h"

#define VOFFSET 5

#define HOFFSET 5

using namespace std;

/\*

\*/

void menuLogIn()

{

doActualizarInventario();

system("cls");

int opt;

string type;

vector<string> options = {"Inicio de sesion", "Registro"};

opt = menu("BIENVENIDO", options);

switch (opt)

{

case 1:

if (!doIniciarSesion(true, type))

menuLogIn();

break;

case 2:

doRegistrarse(false);

menuLogIn();

break;

case 0:

esquinarTexto("Cerrando Programa", 1);

for (int i = 0; i < 3; i++)

{

cout << ".";

cout.flush();

Sleep(1);

}

Sleep(1000);

break;

}

}

void menuMain()

{

doActualizarInventario();

system("cls");

int opt;

vector<string> options = {"Registrar venta", "\"" + userController.getUsuario(progController.getSesionKey()).getUsername() + "\"", "Inventario", "Otros"};

opt = menu("MENU PRINCIPAL", options);

switch (opt)

{

case 1:

doVenta();

menuMain();

break;

case 2:

menuUser();

break;

case 3:

menuInventory();

break;

case 4:

menuOthers();

break;

case 0:

esquinarTexto("-<>-Hasta pronto-<>-", 1);

break;

}

}

void menuUser()

{

doActualizarInventario();

system("cls");

int opt;

vector<string> options = {"Modificar Perfil" /\*, "Registros"\*/, "Registrar Usuario", "Estado de Caja", "Cerrar Sesión"};

opt = menu("MENU USUARIO", options);

switch (opt)

{

case 1:

doModificarPerfil(progController.getSesionKey());

menuUser();

break;

/\*case 2:

menuRecords();

break;\*/

case 2:

doRegistrarse(false);

menuUser();

break;

case 3:

askEstadoCaja();

menuUser();

break;

case 4:

doCerrarSesion();

break;

case 0:

menuMain();

break;

}

}

void menuInventory()

{

doActualizarInventario();

system("cls");

int opt;

vector<string> options = {"Registrar Compra", "Consultar Inventario", "Modificar Datos de producto", "Anadir Nuevo Producto", "Anadir Nueva Marca"};

do

{

opt = menu("MENU INVENTARIO", options);

switch (opt)

{

case 1:

doCompra();

menuInventory();

break;

case 2:

askInventario();

break;

case 3:

changeDataInventario();

break;

case 4:

doAddProducto();

break;

case 5:

doAddMarca();

break;

case 0:

menuMain();

break;

default:

marcaController.saveFile();

break;

}

} while (opt != 0);

}

void menuRecords()

{

doActualizarInventario();

system("cls");

int opt;

vector<string> options = {"Estado de Caja", "Registros Personales", "Buscar Registros"}; // Carritos de compra, etc

do

{

opt = menu("MENU REGISTROS", options);

switch (opt)

{

case 1:

askEstadoCaja();

break;

case 2:

doConsultarRegistro(progController.getSesionKey());

break;

case 3:

doBuscarRegistro();

break;

case 0:

menuMain();

break;

}

} while (opt != 0);

}

void menuOthers()

{

doActualizarInventario();

system("cls");

int opt;

vector<string> options = {"Registrar Movimiento", "Registrar Cliente", "Registrar Proveedor", "Retiro de Caja"}; // Carritos de compra, etc

do

{

opt = menu("MENU REGISTROS", options);

switch (opt)

{

case 1:

doMovimiento();

break;

case 2:

doRegistrarTercero(true, -1);

break;

case 3:

doRegistrarTercero(false, -1);

break;

case 4:

doRegistrarRetiroCaja();

break;

case 0:

menuMain();

break;

}

} while (opt != 0);

}

#endif

##### Funciones.h

#ifndef FUNCIONES\_H

#define FUNCIONES\_H

#include <iostream>

#include <string>

#include "../controller/controllers.h"

#include "../model/FuncionesGenerales.h"

#include "../model/models.h"

using namespace std;

programController progController;

usuarioController userController;

ProductoController productoController;

MarcaController marcaController;

CompraController compraController;

CompraDController compraDController;

VentaController ventaController;

VentaDController ventaDController;

CajaController cajaController;

KardexController kardexController;

ClienteController clienteController;

ProveedorController proveedorController;

// Option Menus

void menuLogIn(); // â

void menuMain(); // â

void menuUser(); // â

void menuInventory(); // â

void menuRecords(); // â

void menuOthers();

// Cuenta y usuario

void doRegistrarse(bool); // â

bool doIniciarSesion(bool, string &); // â

void doCerrarSesion(); // â

void doModificarPerfil(int); // â

void doCompra();

void doVenta();

void doMovimiento();

void askInventario();

void changeDataInventario();

void doAddProducto();

void doAddMarca();

void askEstadoCaja();

void doConsultarRegistro(int);

void doBuscarRegistro();

void doActualizarInventario();

long long int doRegistrarTercero(bool, long long int);

void doRegistrarRetiroCaja();

#endif

#### MODEL/

#ifndef CLASES\_H

#define CLASES\_H

#include <iostream>

#include <string>

#include <vector>

#include "FuncionesGenerales.h"

using namespace std;

class Usuario;

class Procesos;

class Compra;

class Venta;

class ProcesosD;

class CompraD;

class VentaD;

class Terceros;

class Cliente;

class Proveedor;

#include "Usuario.h"

class Componente

{

private:

string nombre;

string cantidad;

public:

Componente();

Componente(string, string);

~Componente();

string getNombre();

string getCantidad();

void setNombre(string);

void setCantidad(string);

};

class Marca

{

private:

int codigoMarca;

string nombreMarca;

public:

Marca();

Marca(int, string);

~Marca();

int getCodigoMarca();

string getNombreMarca();

void setCodigoMarca(int);

void setNombreMarca(string);

};

class Producto

{

private:

int codProducto;

int codMarca;

string nombre;

double precioUnitario;

int stock;

int numComponentes;

vector<Componente> vectorComponentes;

public:

Producto();

Producto(int, int, string, double, int, vector<Componente>);

~Producto();

void setCodProducto(int);

void setCodMarca(int);

void setNombre(string);

void setPrecioUnitario(double);

void setStock(int);

void setNumComponentes(int);

void setComponentes(vector<Componente>);

void modifyStock(int);

int getCodProducto();

int getCodMarca();

string getNombre();

double getPrecioUnitario();

int getStock();

int getNumComponentes();

vector<Componente> getVectorComponentes();

void addComponente(Componente);

void deleteComponente(int);

void deleteMarca(int);

Componente getComponente(int);

};

class Procesos

{

private:

int codUsuario;

string fecha;

double monto;

bool estado;

public:

Procesos();

Procesos(int, double, bool);

~Procesos();

int getCodUsuario();

string getFecha();

double getMonto();

bool getEstado();

void setCodUsuario(int);

void setFecha(string);

void setMonto(double);

void setEstado(bool);

};

class Compra : public Procesos

{

private:

int codCompra;

int codProveedor;

public:

Compra();

Compra(int, int, int, double, bool);

~Compra();

int getCodCompra();

int getCodProveedor();

void setCodCompra(int);

void setCodProveedor(int);

};

class Venta : public Procesos

{

private:

int codVenta;

int codCliente;

public:

Venta();

Venta(int, int, int, double, bool);

~Venta();

int getCodVenta();

int getCodCliente();

void setCodVenta(int);

void setCodCliente(int);

};

class ProcesosD

{

private:

int codProducto;

int cantidad;

double precio;

double monto;

public:

ProcesosD();

ProcesosD(int, int, double);

~ProcesosD();

int getCodProducto();

int getCantidad();

double getPrecio();

double getMonto();

void setCodProducto(int);

void setCantidad(int);

void setPrecio(double);

void setMonto(double);

};

class CompraD : public ProcesosD

{

private:

int codCompra;

public:

CompraD();

CompraD(int, int, int, double);

~CompraD();

int getCodCompra();

void setCodCompra(int);

};

class VentaD : public ProcesosD

{

private:

int codVenta;

public:

VentaD();

VentaD(int, int, int, double);

~VentaD();

int getCodVenta();

void setCodVenta(int);

};

class Terceros

{

private:

string nombre;

string tipoDocumento;

long long int documento;

public:

Terceros();

Terceros(string, string, long long int);

~Terceros();

string getNombre();

string getTipoDocumento();

long long int getDocumento();

void setNombre(string);

void setTipoDocumento(string);

void setDocumento(long long int);

};

class Cliente : public Terceros

{

private:

int codCliente;

public:

Cliente();

Cliente(int, string, long long int);

~Cliente();

void setCodCliente(int);

int getCodCliente();

};

class Proveedor : public Terceros

{

private:

int codProveedor;

public:

Proveedor();

Proveedor(int, string, long long int);

~Proveedor();

void setCodProveedor(int);

int getCodProveedor();

};

#endif // CLASES\_H

#ifndef CLIENTE\_CPP

#define CLIENTE\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

Cliente::Cliente() {}

Cliente::Cliente(int \_codCliente, string \_nombre, long long int \_documento): Terceros(\_nombre, "DNI", \_documento)

{

codCliente = \_codCliente;

}

Cliente::~Cliente() {}

void Cliente::setCodCliente(int \_codCliente)

{

codCliente = \_codCliente;

}

int Cliente::getCodCliente()

{

return codCliente;

}

#endif // CLIENTE\_CPP

#ifndef COMPONENTE\_CPP

#define COMPONENTE\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

Componente::Componente() {}

Componente::Componente(string \_nombre, string \_cantidad)

{

nombre = \_nombre;

cantidad = \_cantidad;

}

Componente::~Componente() {}

string Componente::getNombre()

{

return nombre;

}

string Componente::getCantidad()

{

return cantidad;

}

void Componente::setNombre(string \_nombre)

{

nombre = \_nombre;

}

void Componente::setCantidad(string \_cantidad)

{

cantidad = \_cantidad;

}

#endif // COMPONENTE\_CPP

#ifndef COMPRA\_CPP

#define COMPRA\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

Compra::Compra() {}

Compra::Compra(int \_codCompra, int \_codProveedor, int \_codUsuario, double \_monto, bool \_estado):Procesos(\_codUsuario, \_monto, \_estado)

{

codCompra = \_codCompra;

codProveedor = \_codProveedor;

}

Compra::~Compra() {}

int Compra::getCodCompra()

{

return codCompra;

}

int Compra::getCodProveedor()

{

return codProveedor;

}

void Compra::setCodCompra(int \_codCompra)

{

codCompra = \_codCompra;

}

void Compra::setCodProveedor(int \_codProveedor)

{

codProveedor = \_codProveedor;

}

#endif // COMPRA\_CPP

#ifndef COMPRAD\_CPP

#define COMPRAD\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

CompraD::CompraD() {}

CompraD::CompraD(int \_codCompra, int \_codProducto, int \_cantidad, double \_precio):ProcesosD(\_codProducto, \_cantidad, \_precio)

{

codCompra = \_codCompra;

}

CompraD::~CompraD() {}

int CompraD::getCodCompra()

{

return codCompra;

}

void CompraD::setCodCompra(int \_codCompra)

{

codCompra = \_codCompra;

}

#endif // COMPRAD\_CPP

#ifndef FUNCIONESGENERALES\_H

#define FUNCIONESGENERALES\_H

#include <iostream>

#include <string>

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <conio.h>

#include <vector>

#include <ctime>

#include "../controller/Encrypter/sha256.cpp"

using namespace std;

#define ENTER 13

#define BACKSPACE 8

char baseColor[] = "0f";

string aMayuscula(string);

string aMinuscula(string);

bool esNumero(string);

int subCadenaComunMasLarga(string, string);

float Promediar(float[], int);

float PromediarMayor0(float[], int);

string currentDateTime();

void getValue(string, int \*);

void getValue(string, string \*);

void getValue(string, float \*);

bool confirmar(string, string, int);

void doEndline(int);

void doTab(int);

string doTab(int, string);

string enterContrasena();

void gotoxy(int, int);

void centrarTexto(string, int, bool, bool, int, int);

void alinearXTexto(string, int, bool, int, int, bool);

void dibujarCuadro();

int menu(string, vector<string>);

void esquinarTexto(string, int, bool, bool, bool);

#define XSIZECMD 120

#define YSIZECMD 30

std::string aMayuscula(std::string cadena)

{

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

{

cadena[i] = std::toupper(cadena[i]);

}

return cadena;

}

std::string aMinuscula(std::string cadena)

{

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

{

cadena[i] = std::tolower(cadena[i]);

}

return cadena;

}

bool esNumero(std::string cadena)

{

bool resultado = true, point = false;

int i = 0;

string salida;

int cantidadGuiones = 0, cantidadPuntos = 0;

if (cadena.size() == 0)

return false;

if (cadena.size() == 1 && (cadena[0] == '.' || cadena[0] == '-'))

return false;

for (char x : cadena)

{

if (!isdigit(x) && x != '-' && x != '.')

return false;

if (x == '-')

cantidadGuiones++;

if (x == '.')

cantidadPuntos++;

}

if (cantidadGuiones > 1 || cantidadGuiones > 1)

return false;

if (cadena[i] == '-')

i++;

if (!isdigit(cadena[i]))

return false;

return true;

}

int subCadenaComunMasLarga(std::string cadena1, std::string cadena2)

{

int ans = 0;

for (int i = 0; i < (int)cadena1.length(); i++)

{

for (int j = 0; j < (int)cadena2.length(); j++)

{

int k = 0;

while ((i + k) < (int)cadena1.length() && (j + k) < (int)cadena2.length() && cadena1[i + k] == cadena2[j + k])

{

k = k + 1;

}

ans = std::max(ans, k);

}

}

return ans;

}

float Promediar(float datos[], int cantidadAPromediar)

{

float acumulador = 0, promedio = 0;

for (int i = 0; i < cantidadAPromediar; i++)

{

acumulador = acumulador + datos[i];

}

if (cantidadAPromediar > 0)

promedio = acumulador / cantidadAPromediar;

return promedio;

}

float PromediarMayor0(float datos[], int cantidadDatos)

{

int cantidadAPromediar = 0;

float aPromediar[999], promedio;

for (int i = 0; i < cantidadDatos; i++)

{

if (datos[i] > 0)

{

aPromediar[cantidadAPromediar] = datos[i];

cantidadAPromediar++;

}

}

promedio = Promediar(aPromediar, cantidadAPromediar);

return promedio;

}

string currentDateTime()

{

time\_t t = time(nullptr);

tm \*now = localtime(&t);

char buffer[128];

strftime(buffer, sizeof(buffer), "%d/%m/%Y %X", now);

return buffer;

}

void getValue(string mensaje, string \*dato)

{

cout << mensaje;

cin >> \*dato;

}

void getValue(string mensaje, int \*dato)

{

cout << mensaje;

cin >> \*dato;

}

void getValue(string mensaje, float \*dato)

{

cout << mensaje;

cin >> \*dato;

}

bool confirmar(string message, string message2 = " ", int hoffset = 0)

{

string opt;

cout << endl

<< endl

<< doTab(hoffset, "") << "¿Seguro que desea " << message << "?" << message2 << "(Confirmar: S/s)" << endl;

cout << doTab(hoffset, "") << ">\_ ";

cin >> opt;

if (aMinuscula(opt) == "s")

return true;

else

return false;

}

void doEndline(int i)

{

for (int j = 0; j < i; j++)

{

cout << endl;

}

}

void doTab(int i)

{

for (int j = 0; j < i; j++)

{

cout << "\t";

}

}

string doTab(int i, string entrada)

{

string salida = entrada;

for (int j = 0; j < i; j++)

{

salida = salida + "\t";

}

return salida;

}

#define ENTER 13

#define BACKSPACE 8

string enterContrasena()

{

string password;

char caracter;

caracter = getch();

password = "";

while (caracter != ENTER)

{

if (caracter != BACKSPACE)

{

password.push\_back(caracter);

cout << "\*";

}

else

{

if (password.length() > 0)

{

cout << "\b \b";

password = password.substr(0, password.length() - 1);

}

}

caracter = getch();

}

return password;

}

void gotoxy(int x, int y)

{

HANDLE hCon;

hCon = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD dwPos;

dwPos.X = x;

dwPos.Y = y;

SetConsoleCursorPosition(hCon, dwPos);

}

void centrarTexto(string message, int val = 0, bool horizontal = true, bool vertical = true, int offsetx = 0, int offsety = 0)

{

int x = (XSIZECMD - size(message)) / 2;

int y = YSIZECMD / 2;

if (horizontal && vertical)

{

gotoxy(x + offsetx, y + offsety);

cout << message;

}

else if (horizontal)

{

gotoxy(x + offsetx, val);

cout << message;

}

else if (vertical)

{

gotoxy(val, y + offsety);

cout << message;

}

else

{

gotoxy(val, val);

cout << message;

}

}

void alinearXTexto(string message, int y, bool izquierda = true, int offsetx = 0, int offsety = 0, bool centradoy = false)

{

int valy;

int sizeMessage = size(message);

if (centradoy)

valy = (YSIZECMD / 2) + offsety;

else

valy = y;

if (izquierda)

{

gotoxy(offsetx, valy);

cout << message;

}

else

{

gotoxy(XSIZECMD - offsetx - sizeMessage, valy);

cout << " " << message;

}

}

void dibujarCuadro()

{

for (int i = 0; i < XSIZECMD - 1; i++)

{

string decorador = "-";

gotoxy(i + 1, 1);

cout << decorador;

gotoxy(i + 1, YSIZECMD);

cout << decorador;

}

for (int i = 0; i < YSIZECMD - 1; i++)

{

string decorador = "|";

gotoxy(1, i + 1);

cout << decorador;

gotoxy(XSIZECMD - 1, i + 1);

cout << decorador;

}

// cout << endl

// << endl;

}

void esquinarTexto(string message, int border = 0, bool up = false, bool left = true, bool print = true)

{

int x;

int y;

if (left)

x = border + 1;

else

x = XSIZECMD - 1 - size(message);

if (up)

y = border + 1;

else

y = YSIZECMD - border;

gotoxy(x, y);

if (print)

cout << message;

}

void menuListado(vector<string> messages, int separation = 0, string title = "", bool pausar = true)

{

int sizeY = size(messages) + (size(messages) \* separation) / 2 - 1;

int e = 0;

system("cls");

dibujarCuadro();

if (title != "")

{

sizeY += 2 + separation / 2;

centrarTexto(title, 0, true, true, 0, (-sizeY + e));

e += 2 + separation;

}

for (string x : messages)

{

centrarTexto(x, 0, true, true, 0, (-sizeY + e));

e += 1 + separation;

}

if (pausar)

{

esquinarTexto("", 1, false, true, false);

system("pause");

}

}

void menuError(vector<string> messages, int separation = 0, string color = "0f", string title = "", bool pausar = false)

{

system("color 4f");

menuListado(messages, separation, title, pausar);

if (!pausar)

Sleep(1000);

char inp[] = {'c', 'o', 'l', 'o', 'r', ' ', color[0], color[1], '\0'};

system(inp);

}

void menuDatos(vector<string> messages, vector<string> &answers, int quantCensor = 0, int separation = 0, string title = "")

{

int sizeY = size(messages) + (size(messages) \* separation) / 2 - 1;

int e = 0, f;

system("cls");

dibujarCuadro();

if (title != "")

{

sizeY += 2 + separation / 2;

centrarTexto(title, 0, true, true, 0, (-sizeY + e));

e += 2 + separation;

}

f = e;

for (string x : messages)

{

alinearXTexto(x + ": ", 0, true, 30, (-sizeY + e), true);

e += separation + 1;

}

e = f;

int val = messages.size() - quantCensor;

for (int i = 0; i < messages.size(); i++)

{

string answ;

alinearXTexto(">\_ ", 0, false, 65, (-sizeY + e), true);

if (i < val)

getline(cin, answ);

else

answ = enterContrasena();

e += separation + 1;

answers.push\_back(answ);

}

}

bool menuConfirmar(string message, string message2 = " ")

{

system("cls");

dibujarCuadro();

int sizeY = 3;

int e = 0;

string opt;

centrarTexto("¿" + message + "? " + message2 + ". (Confirmar: S/s)", 0, true, true, 0, -3);

e += 2;

centrarTexto(">\_ ", 0, true, true, -5, 0);

getline(cin,opt);

if (aMinuscula(opt) == "s" || aMinuscula(opt) == "y" || aMinuscula(opt) == "si" || aMinuscula(opt) == "yes")

return true;

else

return false;

}

int menu(string title, vector<string> options)

{

int sizeY = size(options) + 5 - 1;

int e = 0, opcionMayor = 0, salida = 0;

string opc;

do

{

system("cls");

int e = 0, opcionMayor = 0;

dibujarCuadro();

centrarTexto("\_\_\_\_" + title + "\_\_\_\_\_", 0, true, true, 0, (-sizeY + e));

e++;

e++;

for (int i = 0; i < size(options); i++)

{

alinearXTexto(options[i], 0, true, 45, (-sizeY + e), true);

alinearXTexto("[" + to\_string(i + 1) + "]", 0, false, 45, (-sizeY + e), true);

e++;

opcionMayor = i + 1;

}

alinearXTexto("Salir", 0, true, 45, (-sizeY + e), true);

alinearXTexto("[0]", 0, false, 45, (-sizeY + e), true);

e++;

centrarTexto(">\_ ", 0, true, true, -2, (-sizeY + e));

cin >> opc;

if (esNumero(opc))

{

if (stoi(opc) < 0 || stoi(opc) > opcionMayor)

{

vector<string> messages = {"~[[INTRODUZCA UNA OPCION VALIDA]]~"};

menuError(messages);

opc = "-1";

}

else

salida = stoi(opc);

}

else

{

vector<string> messages = {"~[[INTRODUZCA UNA OPCION VALIDA]]~"};

menuError(messages);

opc = "-1";

}

} while (stoi(opc) < 0);

cin.ignore();

return salida;

}

string menuBusqueda(vector<string> \_lista, int separation = 0, string title = "")

{

char opt;

int j;

vector<string> lista = \_lista;

vector<string> similares;

vector<string> similares2;

string busqueda = "";

string busquedaString;

string busquedaLista;

do

{

int sizeY = 1 + (1 \* separation) / 2 - 1;

int e = 0;

system("cls");

dibujarCuadro();

j = 1;

if (title != "")

{

sizeY += 2 + separation / 2;

centrarTexto(title, 0, true, true, 0, (-sizeY + e));

e += 2 + separation;

}

centrarTexto(">\_" + busqueda, 0, true, true, 0, (-sizeY + e));

e += 2 + separation;

for (string x:similares)

{

if (j <= 9)

{

while (x.size() < 15)

x.push\_back(' ');

centrarTexto("[" + to\_string(j) + "] " + x, 0, true, true, 0, (-sizeY + e));

j++;

e += 1 + separation;

}

}

centrarTexto("[0] Salir ", 0, true, true, 0, (-sizeY + e));

e += 1 + separation;

gotoxy(61 + (busqueda.size() / 2), 15);

opt = getch();

switch (opt)

{

case '\b':

if (busqueda.size() != 0)

{

busqueda.erase(busqueda.begin() + busqueda.size() - 1);

similares.clear();

for(int i = 0; i < lista.size(); i++)

{

busquedaLista = "";

busquedaString = aMinuscula(busqueda);

for (int k = 0; k < busqueda.size(); k++)

{

string h = aMinuscula(lista[i]);

busquedaLista.push\_back(h[k]);

}

if (busquedaString == busquedaLista)

similares.push\_back(lista[i]);

}

}

break;

default:

if (opt == '0')

break;

if (isdigit(opt))

{

if (similares.size() != 0)

return similares[int(opt) - 49];

else

break;

}

busqueda.push\_back(opt);

if (similares.size() == 0)

for(int i = 0; i < lista.size(); i++)

{

busquedaLista = "";

busquedaString = aMinuscula(busqueda);

for (int k = 0; k < busqueda.size(); k++)

{

string h = aMinuscula(lista[i]);

busquedaLista.push\_back(h[k]);

}

if (busquedaString == busquedaLista)

similares.push\_back(lista[i]);

}

else

{

for(int i = 0; i < similares.size(); i++)

{

busquedaLista = "";

busquedaString = aMinuscula(busqueda);

for (int k = 0; k < busqueda.size(); k++)

{

string h = aMinuscula(similares[i]);

busquedaLista.push\_back(h[k]);

}

if (busquedaString == busquedaLista)

similares2.push\_back(similares[i]);

}

similares = similares2;

similares2.clear();

}

break;

}

} while(opt != '0');

return "salir";

}

string createCode(string start,int correlativo, int cantidadMinimaDigitos=0)

{

string salida=start;

for (int i = to\_string(correlativo).size(); i < cantidadMinimaDigitos; i++)

salida.push\_back('0');

salida = salida + to\_string(correlativo);

return salida;

}

#endif

#ifndef KARDEX\_H

#define KARDEX\_H

#include <iostream>

#include <string>

using namespace std;

/\*

\*/

class kardex

{

private:

string fechaDeEmision;

int cantidad;

double montoUnitario;

int codProducto;

string codProceso;

string comprobante;

bool isSalida;

string motivo;

string observacion;

public:

kardex();

kardex(string, int, int,double, string, string, bool, string, string);

~kardex();

string getFechaDeEmision();

int getCantidad();

double getMontoUnitario();

int getCodProducto();

string getCodProceso();

string getComprobante();

bool getIsSalida();

string getMotivo();

string getObservacion();

void setFechaDeEmision(string);

void setCantidad(int);

void setMontoUnitario(double);

void setCodProducto(int);

void setCodProceso(string);

void setComprobante(string);

void setIsSalida(bool);

void setMotivo(string);

void setObservacion(string);

};

#endif

#ifndef KARDEX\_CPP

#define KARDEX\_CPP

#include <iostream>

#include <string>

#include <vector>

#include "kardex.h"

#include "FuncionesGenerales.h"

using namespace std;

kardex::kardex()

{

}

kardex::kardex(string \_fechaDeEmision, int \_cantidad, int \_codProducto, double \_montoUnitario, string \_codProceso, string \_comprobante, bool \_isSalida, string \_motivo, string \_observacion)

{

fechaDeEmision = \_fechaDeEmision;

cantidad = \_cantidad;

montoUnitario=\_montoUnitario;

codProducto = \_codProducto;

codProceso = \_codProceso;

comprobante = \_comprobante;

isSalida = \_isSalida;

motivo = \_motivo;

observacion = \_observacion;

}

kardex::~kardex()

{

}

string kardex::getFechaDeEmision() { return fechaDeEmision; }

int kardex::getCantidad() { return cantidad; }

double kardex::getMontoUnitario() { return montoUnitario; }

int kardex::getCodProducto() { return codProducto; }

string kardex::getCodProceso() { return codProceso; }

string kardex::getComprobante() { return comprobante; }

bool kardex::getIsSalida() { return isSalida; }

string kardex::getMotivo() { return motivo; }

string kardex::getObservacion() { return observacion; }

void kardex::setFechaDeEmision(string \_fechaDeEmision) { fechaDeEmision = \_fechaDeEmision; }

void kardex::setCantidad(int \_cantidad) { cantidad = \_cantidad; }

void kardex::setMontoUnitario(double \_montoUnitario){montoUnitario=\_montoUnitario;}

void kardex::setCodProducto(int \_codProducto) { codProducto = \_codProducto; }

void kardex::setCodProceso(string \_codProceso) { codProceso = \_codProceso; }

void kardex::setComprobante(string \_comprobante) { comprobante = \_comprobante; }

void kardex::setIsSalida(bool \_isSalida) { isSalida = \_isSalida; }

void kardex::setMotivo(string \_motivo) { motivo = \_motivo; }

void kardex::setObservacion(string \_observacion) { observacion = \_observacion; }

#endif

#ifndef MARCA\_H

#define MARCA\_H

#include <iostream>

#include <string>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

Marca::Marca() {}

Marca::Marca(int cod, string \_nombreMarca)

{

codigoMarca = cod;

nombreMarca = \_nombreMarca;

}

Marca::~Marca() {}

int Marca::getCodigoMarca()

{

return codigoMarca;

}

string Marca::getNombreMarca()

{

return nombreMarca;

}

void Marca::setCodigoMarca(int cod)

{

codigoMarca = cod;

}

void Marca::setNombreMarca(string \_nombreMarca)

{

nombreMarca = \_nombreMarca;

}

#endif // MARCA\_H

#ifndef MODELS\_H

#define MODELS\_H

#include "Procesos.cpp"

#include "ProcesosD.cpp"

#include "Terceros.cpp"

#endif

#ifndef PROCESOS\_CPP

#define PROCESOS\_CPP

#include <iostream>

#include <string>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

Procesos::Procesos() {}

Procesos::Procesos(int \_codUsuario, double \_monto, bool \_estado)

{

codUsuario = \_codUsuario;

monto = \_monto;

estado = \_estado;

fecha = currentDateTime();

}

Procesos::~Procesos() {}

int Procesos::getCodUsuario()

{

return codUsuario;

}

string Procesos::getFecha()

{

return fecha;

}

double Procesos::getMonto()

{

return monto;

}

bool Procesos::getEstado()

{

return estado;

}

void Procesos::setCodUsuario(int \_codUsuario)

{

codUsuario = \_codUsuario;

}

void Procesos::setFecha(string \_fecha)

{

fecha = \_fecha;

}

void Procesos::setMonto(double \_monto)

{

monto = \_monto;

}

void Procesos::setEstado(bool \_estado)

{

estado = \_estado;

}

#endif // PROCESOS\_CPP

#ifndef PROCESOSD\_CPP

#define PROCESOSD\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

ProcesosD::ProcesosD() {}

ProcesosD::ProcesosD(int \_codProducto, int \_cantidad, double \_precio)

{

codProducto = \_codProducto;

cantidad = \_cantidad;

precio = \_precio;

monto = precio \* cantidad;

}

ProcesosD::~ProcesosD() {}

int ProcesosD::getCodProducto()

{

return codProducto;

}

int ProcesosD::getCantidad()

{

return cantidad;

}

double ProcesosD::getPrecio()

{

return precio;

}

double ProcesosD::getMonto()

{

return monto;

}

void ProcesosD::setCodProducto(int \_codProducto)

{

codProducto = \_codProducto;

}

void ProcesosD::setCantidad(int \_cantidad)

{

cantidad = \_cantidad;

}

void ProcesosD::setPrecio(double \_precio)

{

precio = \_precio;

}

void ProcesosD::setMonto(double \_monto)

{

monto = \_monto;

}

#endif // PROCESOSD\_CPP

#ifndef PRODUCTO\_CPP

#define PRODUCTO\_CPP

#include <iostream>

#include <string>

#include <vector>

#include <random>

#include "clases.h"

#include "FuncionesGenerales.h"

#include "Componente.cpp"

#include "Marca.cpp"

Producto::Producto() {}

Producto::Producto(int codProducto, int codMarca, string nombre, double precioUnitario, int stock, vector<Componente> vectorComponentes)

{

this->codProducto = codProducto;

this->codMarca = codMarca;

this->nombre = nombre;

this->precioUnitario = precioUnitario;

this->stock = stock;

this->numComponentes = vectorComponentes.size();

this->vectorComponentes = vectorComponentes;

}

Producto::~Producto() {}

void Producto::setCodProducto(int codProducto) { this->codProducto = codProducto; }

void Producto::setCodMarca(int codMarca) { this->codMarca = codMarca; }

void Producto::setNombre(string nombre) { this->nombre = nombre; }

void Producto::setPrecioUnitario(double precioUnitario) { this->precioUnitario = precioUnitario; }

void Producto::setStock(int stock) { this->stock = stock; }

void Producto::setNumComponentes(int numComponentes) { this->numComponentes = numComponentes; }

void Producto::setComponentes(vector<Componente> vector)

{

vectorComponentes = vector;

setNumComponentes(vectorComponentes.size());

}

void Producto::modifyStock(int value = 1) { stock += value; }

int Producto::getCodProducto() { return codProducto; }

int Producto::getCodMarca() { return codMarca; }

string Producto::getNombre() { return nombre; }

double Producto::getPrecioUnitario() { return precioUnitario; }

int Producto::getStock() { return stock; }

int Producto::getNumComponentes() { return numComponentes; }

vector<Componente> Producto::getVectorComponentes() { return vectorComponentes; }

void Producto::addComponente(Componente comp) { vectorComponentes.push\_back(comp); }

void Producto::deleteComponente(int pos) { vectorComponentes.erase(vectorComponentes.begin() + pos); }

Componente Producto::getComponente(int pos) { return vectorComponentes[pos]; }

#endif // PRODUCTO\_CPP

#ifndef PROVEEDOR\_CPP

#define PROVEEDOR\_CPP

#include <iostream>

#include "clases.h"

#include "../model/FuncionesGenerales.h"

Proveedor::Proveedor() {}

Proveedor::Proveedor(int \_codProveedor, string \_nombre, long long int \_documento): Terceros(\_nombre, "RUC", \_documento)

{

codProveedor = \_codProveedor;

}

Proveedor::~Proveedor() {}

void Proveedor::setCodProveedor(int \_codProveedor)

{

codProveedor = \_codProveedor;

}

int Proveedor::getCodProveedor()

{

return codProveedor;

}

#endif // PROVEEDOR\_CPP

#ifndef TERCEROS\_CPP

#define TERCEROS\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

Terceros::Terceros() {}

Terceros::Terceros(string \_nombre, string \_tipoDocumento, long long int \_documento)

{

nombre = \_nombre;

tipoDocumento = \_tipoDocumento;

documento = \_documento;

}

Terceros::~Terceros() {}

string Terceros::getNombre()

{

return nombre;

}

string Terceros::getTipoDocumento()

{

return tipoDocumento;

}

long long int Terceros::getDocumento()

{

return documento;

}

void Terceros::setNombre(string \_nombre)

{

nombre = \_nombre;

}

void Terceros::setTipoDocumento(string \_tipoDocumento)

{

tipoDocumento = \_tipoDocumento;

}

void Terceros::setDocumento(long long int \_documento)

{

documento = \_documento;

}

#endif // TERCEROS\_CPP

#ifndef USUARIO\_CPP

#define USUARIO\_CPP

#include <iostream>

#include <string>

#include <vector>

#include <random>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

using namespace std;

Usuario::Usuario() {}

Usuario::~Usuario() {}

Usuario::Usuario(int \_codUsuario,string \_username, string \_contrasena, string \_nombre, string \_apellidos, string \_tipoDocumento, int \_numDocumento, string \_tipoUsuario)

{

codUsuario=\_codUsuario;

username = \_username;

contrasena = \_contrasena;

nombre = \_nombre;

apellidos = \_apellidos;

tipoDocumento = \_tipoDocumento;

numDocumento = \_numDocumento;

tipoUsuario = \_tipoUsuario;

}

void Usuario::setCodUsuario(int \_codUsuario)

{

codUsuario=\_codUsuario;

}

void Usuario::setUsername(string \_username)

{

username = \_username;

}

void Usuario::setNombre(string \_nombre)

{

nombre = \_nombre;

}

void Usuario::setApellidos(string \_apellidos)

{

apellidos = \_apellidos;

}

void Usuario::setDocumento(string \_tipoDocumento)

{

tipoDocumento = \_tipoDocumento;

}

void Usuario::setNumDocumento(int \_numDocumento)

{

numDocumento = \_numDocumento;

}

void Usuario::setContrasena(string \_contrasena)

{

contrasena = \_contrasena;

}

void Usuario::setTipoUsuario(string \_tipoUsuario)

{

tipoUsuario = \_tipoUsuario;

}

int Usuario::getCodUsuario()

{

return codUsuario;

}

string Usuario::getUsername()

{

return username;

}

string Usuario::getNombre()

{

return nombre;

}

string Usuario::getApellidos()

{

return apellidos;

}

string Usuario::getTipoDocumento()

{

return tipoDocumento;

}

int Usuario::getNumDocumento()

{

return numDocumento;

}

string Usuario::getContrasena()

{

return contrasena;

}

string Usuario::getTipoUsuario()

{

return tipoUsuario;

}

void Usuario::listarDatos()

{

system("cls");

cout<<"Codigo: "<<getCodUsuario()<<endl;

cout << "Usuario: " << getUsername() << endl;

cout << "ContraseÃ±a: " << getContrasena() << endl;

cout << "Nombre: " << getNombre() << endl;

cout << "Apellidos: " << getApellidos() << endl;

cout << "Tipo de usuario: " << getTipoUsuario() << endl;

cout << "Tipo de documento: " << getTipoDocumento() << endl;

cout << "Numero de documento: " << getNumDocumento() << endl;

system("pause");

}

#endif

#ifndef USUARIO\_H

#define USUARIO\_H

#include <iostream>

#include <string>

#include <vector>

using namespace std;

/\*

\*/

class Usuario

{

private:

int codUsuario;

string username;

string nombre;

string apellidos;

string tipoDocumento;

long int numDocumento;

string contrasena;

string tipoUsuario;

public:

Usuario();

~Usuario();

Usuario(int, string, string, string, string, string, int, string);

void setCodUsuario(int);

void setUsername(string);

void setNombre(string);

void setApellidos(string);

void setDocumento(string);

void setNumDocumento(int);

void setContrasena(string);

void setTipoUsuario(string);

int getCodUsuario();

string getUsername();

string getNombre();

string getApellidos();

string getTipoDocumento();

int getNumDocumento();

string getContrasena();

string getTipoUsuario();

void listarDatos();

};

#endif

#ifndef VENTA\_CPP

#define VENTA\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

Venta::Venta() {}

Venta::Venta(int \_codVenta, int \_codCliente, int \_codUsuario, double \_monto, bool \_estado):Procesos(\_codUsuario, \_monto, \_estado)

{

codVenta = \_codVenta;

codCliente = \_codCliente;

}

Venta::~Venta() {}

int Venta::getCodVenta()

{

return codVenta;

}

int Venta::getCodCliente()

{

return codCliente;

}

void Venta::setCodVenta(int \_codVenta)

{

codVenta = \_codVenta;

}

void Venta::setCodCliente(int \_codCliente)

{

codCliente = \_codCliente;

}

#endif // VENTA\_CPP

#ifndef VENTAD\_CPP

#define VENTAD\_CPP

#include <iostream>

#include "Clases.h"

#include "../model/FuncionesGenerales.h"

VentaD::VentaD() {}

VentaD::VentaD(int \_codVenta, int \_codProducto, int \_cantidad, double \_precio):ProcesosD(\_codProducto, \_cantidad, \_precio)

{

codVenta = \_codVenta;

}

VentaD::~VentaD() {}

int VentaD::getCodVenta()

{

return codVenta;

}

void VentaD::setCodVenta(int \_codVenta)

{

codVenta = \_codVenta;

}

#endif // VENTAD\_CPP

#### CONTROLLER/

#ifndef CAJACONTROLLER\_H

#define CAJACONTROLLER\_H

#include <iostream>

class CajaController

{

private:

double saldo;

public:

CajaController();

CajaController(double);

~CajaController();

void setSaldo(double);

double getSaldo();

void modifySaldo(double, bool);

};

CajaController::CajaController() {}

CajaController::CajaController(double \_saldo)

{

saldo = \_saldo;

}

CajaController::~CajaController() {}

void CajaController::setSaldo(double \_saldo)

{

saldo = \_saldo;

}

double CajaController::getSaldo()

{

return saldo;

}

void CajaController::modifySaldo(double monto, bool operacion)

{

if (operacion)

saldo += monto;

else

saldo -= monto;

}

#endif // CAJACONTROLLER\_H

#ifndef CLIENTECONTROLLER\_H

#define CLIENTECONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/Cliente.cpp"

class ClienteController

{

private:

vector<Cliente> vectorCliente;

public:

ClienteController();

int getCorrelativo();

void add(Cliente);

void modify(Cliente, int);

int size();

Cliente get(int);

int getPosDoc(long long int,string);

void saveFile();

void copyFile();

};

ClienteController::ClienteController()

{

copyFile();

}

int ClienteController::getCorrelativo()

{

if(size()==0)

{

return 1;

}

else

{

return vectorCliente[size() - 1].getCodCliente() + 1;

}

}

void ClienteController::add(Cliente obj)

{

vectorCliente.push\_back(obj);

}

void ClienteController::modify(Cliente temp, int obj)

{

vectorCliente[obj] = temp;

}

int ClienteController::size()

{

return vectorCliente.size();

}

Cliente ClienteController::get(int pos)

{

return vectorCliente[pos];

}

int ClienteController::getPosDoc(long long int Doc,string typeDoc)

{

for (Cliente x : vectorCliente)

{

if(x.getDocumento()==Doc&&x.getTipoDocumento()==typeDoc)

return x.getCodCliente();

}

return -1;

}

void ClienteController::saveFile()

{

try

{

fstream archivoClientes;

archivoClientes.open("../data/clientes.csv", ios::app);

if (archivoClientes.is\_open())

{

Cliente obj = vectorCliente.back();

archivoClientes << obj.getCodCliente() << ","

<< obj.getNombre() << ","

<< obj.getTipoDocumento() << ","

<< obj.getDocumento() << ",";

archivoClientes << endl;

archivoClientes.close();

}

}

catch(exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void ClienteController::copyFile()

{

try

{

int i;

size\_t posi;

string linea;

fstream archivoClientes;

vectorCliente.clear();

archivoClientes.open("../data/clientes.csv", ios::in);

if (archivoClientes.is\_open())

{

while (!archivoClientes.eof() && getline(archivoClientes, linea))

{

vector<string> temporal;

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

Cliente obj;

obj.setCodCliente(stoi(temporal[0]));

obj.setNombre(temporal[1]);

obj.setTipoDocumento(temporal[2]);

obj.setDocumento(stoll(temporal[3]));

add(obj);

}

archivoClientes.close();

}

}

catch(exception e)

{

cout << "Ocurrio un error al leer el archivo";

}

}

#endif // CLIENTECONTROLLER\_H

#ifndef COMPRACONTROLLER\_H

#define COMPRACONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/Compra.cpp"

using namespace std;

class CompraController

{

private:

vector<Compra> vectorCompra;

public:

CompraController();

void add(Compra);

int getCorrelativo();

void modify(Compra, int);

int size();

Compra get(int);

void saveFile();

void copyFile();

};

CompraController::CompraController()

{

copyFile();

}

void CompraController::add(Compra obj)

{

vectorCompra.push\_back(obj);

}

int CompraController::getCorrelativo()

{

if(size() == 0)

{

return 1;

}

else

{

return vectorCompra[size() - 1].getCodCompra() + 1;

}

}

void CompraController::modify(Compra temp, int obj)

{

vectorCompra[obj] = temp;

}

int CompraController::size()

{

return vectorCompra.size();

}

Compra CompraController::get(int pos)

{

return vectorCompra[pos];

}

void CompraController::saveFile()

{

try

{

fstream archivoCompras;

archivoCompras.open("../data/compras.csv", ios::app);

if (archivoCompras.is\_open())

{

Compra obj = vectorCompra.back();

archivoCompras << obj.getCodCompra() << ","

<< obj.getCodProveedor() << ","

<< obj.getCodUsuario() << ","

<< obj.getEstado() << ","

<< obj.getFecha() << ","

<< obj.getMonto() << ","

<< endl;

archivoCompras.close();

}

}

catch(exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void CompraController::copyFile()

{

try

{

int i;

size\_t posi;

string linea;

fstream archivoCompras;

vectorCompra.clear();

archivoCompras.open("../data/compras.csv", ios::in);

if (archivoCompras.is\_open())

{

while (!archivoCompras.eof() && getline(archivoCompras, linea))

{

vector<string> temporal;

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

Compra obj;

obj.setCodCompra(stoi(temporal[0]));

obj.setCodProveedor(stoi(temporal[1]));

obj.setCodUsuario(stoi(temporal[2]));

if (temporal[3] == "true")

{

obj.setEstado(true);

}

else

{

obj.setEstado(false);

}

obj.setFecha(temporal[4]);

obj.setMonto(stod(temporal[5]));

add(obj);

}

archivoCompras.close();

}

}

catch(exception e)

{

cout << "Ocurrio un error al leer el archivo";

system("pause");

}

}

#endif // COMPRACONTROLLER\_H

#ifndef COMPRADCONTROLLER\_H

#define COMPRADCONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/CompraD.cpp"

using namespace std;

class CompraDController

{

private:

vector<CompraD> vectorCompraD;

public:

CompraDController();

void add(CompraD);

void modify(CompraD, int);

int size();

CompraD get(int);

void saveFile();

void copyFile();

};

CompraDController::CompraDController()

{

copyFile();

}

void CompraDController::add(CompraD obj)

{

vectorCompraD.push\_back(obj);

}

void CompraDController::modify(CompraD temp, int obj)

{

vectorCompraD[obj] = temp;

}

int CompraDController::size()

{

return vectorCompraD.size();

}

CompraD CompraDController::get(int pos)

{

return vectorCompraD[pos];

}

void CompraDController::saveFile()

{

try

{

fstream archivoComprasD;

archivoComprasD.open("../data/comprasD.csv", ios::out);

if (archivoComprasD.is\_open())

{

CompraD obj = vectorCompraD.back();

archivoComprasD << obj.getCodCompra() << ","

<< obj.getCodProducto() << ","

<< obj.getCantidad() << ","

<< obj.getPrecio() << ","

<< obj.getMonto() << ","

<< endl;

archivoComprasD.close();

}

}

catch(exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void CompraDController::copyFile()

{

try

{

int i;

size\_t posi;

string linea;

fstream archivoComprasD;

vectorCompraD.clear();

archivoComprasD.open("../data/comprasD.csv", ios::in);

if (archivoComprasD.is\_open())

{

while (!archivoComprasD.eof() && getline(archivoComprasD, linea))

{

vector<string> temporal;

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

CompraD obj;

obj.setCodCompra(stoi(temporal[0]));

obj.setCodProducto(stoi(temporal[1]));

obj.setCantidad(stoi(temporal[2]));

obj.setPrecio(stod(temporal[3]));

obj.setMonto(stod(temporal[4]));

add(obj);

}

archivoComprasD.close();

}

}

catch(exception e)

{

cout << "Ocurrio un error al leer el archivo";

system("pause");

}

}

#endif // COMPRADCONTROLLER\_H

#ifndef CONTROLLERS\_H

#define CONTROLLERS\_H

#include "usuarioController.h"

#include "programController.h"

#include "marcaController.h"

#include "productoController.h"

#include "compraController.h"

#include "compraDController.h"

#include "ventaController.h"

#include "ventaDController.h"

#include "cajaController.h"

#include "clienteController.h"

#include "proveedorController.h"

#include "compraDController.h"

#include "compraDController.h"

#include "KardexController.h"

#endif

#ifndef KARDEXCONTROLLER\_H

#define KARDEXCONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/kardex.cpp"

using namespace std;

/\*

\*/

class KardexController

{

private:

vector<kardex> vectorKardex;

public:

KardexController();

void add(kardex);

void modify(kardex, int);

kardex get(int);

vector<kardex> getVector();

void saveFile();

void saveFileAll();

void getFile();

bool validarFormatoFecha(string);

vector<kardex> getMovimientosProducto(int);

vector<kardex> getMovimientosComprobante(bool, bool);

vector<kardex> getMovimientosRetiros();

vector<kardex> getMovimientosProducto(int, bool);

int getEstadoCaja();

int getCantidadComprobanteVenta(bool);

int getCantidadComprobante(bool, bool);

int getCantidadProducto(int);

};

KardexController::KardexController() { getFile(); }

void KardexController::add(kardex obj) { vectorKardex.push\_back(obj); }

void KardexController::modify(kardex obj, int pos) { vectorKardex[pos] = obj; }

kardex KardexController::get(int pos) { return vectorKardex[pos]; }

vector<kardex> KardexController::getVector() { return vectorKardex; }

void KardexController::saveFile()

{

try

{

fstream archivoKardex;

archivoKardex.open("../data/movimientos.csv", ios::app);

if (archivoKardex.is\_open())

{

kardex obj = vectorKardex.back();

string isSalida = obj.getIsSalida() ? "true" : "false";

archivoKardex << obj.getFechaDeEmision() << ","

<< obj.getCantidad() << ","

<< obj.getCodProducto() << ","

<< obj.getCodProceso() << ","

<< obj.getComprobante() << ","

<< isSalida << ","

<< obj.getMotivo() << ","

<< obj.getObservacion() << ","

<< obj.getMontoUnitario() << ",";

archivoKardex << endl;

}

archivoKardex.close();

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void KardexController::saveFileAll()

{

try

{

fstream archivoKardex;

archivoKardex.open("../data/movimientos.csv", ios::out);

if (archivoKardex.is\_open())

{

for (kardex obj : vectorKardex)

{

string isSalida = obj.getIsSalida() ? "true" : "false";

archivoKardex << obj.getFechaDeEmision() << ","

<< obj.getCantidad() << ","

<< obj.getCodProducto() << ","

<< obj.getCodProceso() << ","

<< obj.getComprobante() << ","

<< isSalida << ","

<< obj.getMotivo() << ","

<< obj.getObservacion() << ","

<< obj.getMontoUnitario() << ",";

archivoKardex << endl;

}

}

archivoKardex.close();

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void KardexController::getFile()

{

int i;

size\_t posi; // Cantidad maxima

string linea;

vector<string> temporal; // Cantidad de columnas

fstream archivoKardex;

archivoKardex.open("../data/movimientos.csv", ios::in);

if (archivoKardex.is\_open())

{

while (!archivoKardex.eof() && getline(archivoKardex, linea))

{

temporal.clear();

i = 0;

while ((posi = linea.find(",")) != string::npos)

{ /\*string::npos es -1, termina cuando llega a este punto\*/

temporal.push\_back(linea.substr(0, posi)); /\*posi = Es la cantidad de caracteres antes del ;\*/

linea.erase(0, posi + 1); // borra la palabra de la primera posiciï¿œn encontrada y con el +1 incluye hasta el ; y luego borra ese elemento, para que en la siguiente iteracion iniciar con la siguiente palabra y de ese modo seguir el proceso ,

i++;

}

// Asignando los valores al vector

bool isSalida = (temporal[5] == "true");

kardex objKardex(temporal[0], stoi(temporal[1]), stoi(temporal[2]), stoi(temporal[8]), temporal[3], temporal[4], isSalida, temporal[6], temporal[7]);

cout << "Movimiento " << temporal[0] << " cargado..." << endl;

Sleep(1);

add(objKardex);

}

}

archivoKardex.close();

}

bool KardexController::validarFormatoFecha(string entrada)

{

size\_t posi;

int i = 0;

vector<string> temporal;

int e = 0;

entrada.push\_back('/');

while ((posi = entrada.find("/")) != string::npos)

{ /\*string::npos es -1, termina cuando llega a este punto\*/

temporal.push\_back(entrada.substr(0, posi)); /\*posi = Es la cantidad de caracteres antes del ;\*/

entrada.erase(0, posi + 1); // borra la palabra de la primera posiciï¿œn encontrada y con el +1 incluye hasta el ; y luego borra ese elemento, para que en la siguiente iteracion iniciar con la siguiente palabra y de ese modo seguir el proceso ,

i++;

}

if (temporal.size() != 3)

return false;

if (temporal[0].size() != 2 || temporal[1].size() != 2 || temporal[2].size() != 4)

return false;

if (!esNumero(temporal[0]) || !esNumero(temporal[1]) || !esNumero(temporal[2]))

return false;

if ((stoi(temporal[0]) < 1 || stoi(temporal[0]) > 31) || (stoi(temporal[1]) < 1 || stoi(temporal[1]) > 12))

return false;

if (!(stoi(temporal[1]) == 1 || stoi(temporal[1]) == 3 || stoi(temporal[1]) == 5 || stoi(temporal[1]) == 7 || stoi(temporal[1]) == 8 || stoi(temporal[1]) == 10 || stoi(temporal[1]) == 12))

{

if (stoi(temporal[0]) > 30)

return false;

if (stoi(temporal[1]) == 2)

{

if (stoi(temporal[0]) > 29)

return false;

if (!((stoi(temporal[2]) % 4 == 0 && stoi(temporal[2]) % 100 != 0) || stoi(temporal[2]) % 400 == 0))

if (stoi(temporal[0]) > 28)

return false;

}

}

return true;

}

vector<kardex> KardexController::getMovimientosProducto(int cod)

{

vector<kardex> salida;

for (kardex x : vectorKardex)

{

if (x.getCodProducto() == cod)

salida.push\_back(x);

}

return salida;

}

vector<kardex> KardexController::getMovimientosProducto(int cod, bool isSalida)

{

vector<kardex> salida;

vector<kardex> movimientos;

movimientos = getMovimientosProducto(cod);

for (kardex x : movimientos)

{

if (x.getIsSalida() == isSalida)

salida.push\_back(x);

}

}

int KardexController::getEstadoCaja()

{

int salida = 0;

vector<double> suma, resta;

for (kardex x : vectorKardex)

{

if(x.getCodProceso().substr(0, 2) == "VT")

suma.push\_back(x.getMontoUnitario()\*x.getCantidad());

else if(x.getCodProceso().substr(0, 2) == "RT")

resta.push\_back(x.getMontoUnitario()\*x.getCantidad());

}

for (double x : suma)

salida += x;

for (double x : resta)

salida -= x;

return salida;

}

vector<kardex> KardexController::getMovimientosRetiros()

{

vector<kardex> salida;

string iniciales ="RT";

for (kardex x : vectorKardex)

{

if (x.getCodProceso().substr(0, 2) == iniciales)

salida.push\_back(x);

}

return salida;

}

vector<kardex> KardexController::getMovimientosComprobante(bool isVenta, bool isMovimiento = false)

{

vector<kardex> salida;

string iniciales = isMovimiento ? "MT" : isVenta ? "VT"

: "CP";

for (kardex x : vectorKardex)

{

if (x.getCodProceso().substr(0, 2) == iniciales)

salida.push\_back(x);

}

return salida;

}

int KardexController::getCantidadComprobanteVenta(bool isBoleta)

{

vector<kardex> salida, entrada;

string iniciales = isBoleta ? "BV" : "FV";

entrada = getMovimientosComprobante(true);

for (kardex x : entrada)

{

if (x.getComprobante().substr(0, 2) == iniciales)

salida.push\_back(x);

}

return salida.size();

}

int KardexController::getCantidadComprobante(bool isVenta, bool isMovimiento = false)

{

vector<kardex> comprobantes;

comprobantes = getMovimientosComprobante(isVenta, isMovimiento);

return comprobantes.size();

}

int KardexController::getCantidadProducto(int cod)

{

int salida=0;

vector<int> aSumar, aRestar;

vector<kardex> movimientos;

movimientos = getMovimientosProducto(cod);

for (kardex x : movimientos)

{

if (x.getIsSalida())

aRestar.push\_back(x.getCantidad());

else

aSumar.push\_back(x.getCantidad());

}

for (int x : aSumar)

salida += x;

for (int x : aRestar)

salida -= x;

return salida;

}

#endif

#ifndef MARCACONTROLLER\_H

#define MARCACONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/Marca.cpp"

class MarcaController

{

private:

vector<Marca> vectorMarca;

public:

MarcaController();

void add(Marca);

Marca get(int);

void modify(Marca, int);

int getNewCodMarca();

bool marcaRegistrada(string);

void saveFile();

void getFile();

};

MarcaController::MarcaController() { getFile(); }

void MarcaController::add(Marca obj) { vectorMarca.push\_back(obj); }

Marca MarcaController::get(int pos) { return vectorMarca[pos]; }

void MarcaController::modify(Marca temp, int obj) { vectorMarca[obj] = temp; }

int MarcaController::getNewCodMarca() { return vectorMarca.size(); }

bool MarcaController::marcaRegistrada(string \_nombre)

{

for (Marca x : vectorMarca)

{

if (aMayuscula(x.getNombreMarca()) == aMayuscula(\_nombre))

return true;

}

return false;

}

void MarcaController::saveFile()

{

try

{

fstream archivoMarcas;

archivoMarcas.open("../data/marcas.csv", ios::out);

if (archivoMarcas.is\_open())

{

for (Marca obj : vectorMarca)

{

archivoMarcas << obj.getCodigoMarca() << "," << obj.getNombreMarca() << ","<<endl;

}

}

archivoMarcas.close();

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void MarcaController::getFile()

{

try

{

int i;

size\_t posi;

string linea;

fstream archivoMarcas;

vectorMarca.clear();

archivoMarcas.open("../data/marcas.csv", ios::in);

if (archivoMarcas.is\_open())

{

while (!archivoMarcas.eof() && getline(archivoMarcas, linea))

{

vector<string> temporal;

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

Marca obj(stoi(temporal[0]),temporal[1]);

add(obj);

}

archivoMarcas.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al leer el archivo";

}

}

#endif // PRODUCTOCONTROLLER\_H

#ifndef PRODUCTOCONTROLLER\_H

#define PRODUCTOCONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/Producto.cpp"

using namespace std;

class ProductoController

{

private:

vector<Producto> vectorProducto;

public:

ProductoController();

void add(Producto);

Producto get(int);

int size();

void modify(Producto, int);

int getNewCodProducto();

bool nombreRegistrado(string);

bool codigoRegistrado(int);

void saveFile();

void getFile();

};

ProductoController::ProductoController() { getFile(); }

void ProductoController::add(Producto obj) { vectorProducto.push\_back(obj); }

Producto ProductoController::get(int pos) { return vectorProducto[pos]; }

int ProductoController::size() { return vectorProducto.size(); }

void ProductoController::modify(Producto temp, int obj) { vectorProducto[obj] = temp; }

int ProductoController::getNewCodProducto() { return vectorProducto.size(); }

bool ProductoController::nombreRegistrado(string \_nombre)

{

for (Producto x : vectorProducto)

{

if (aMayuscula(x.getNombre()) == aMayuscula(\_nombre))

return true;

}

return false;

}

bool ProductoController::codigoRegistrado(int \_cod)

{

for (Producto x : vectorProducto)

{

if (\_cod == x.getCodProducto())

return true;

}

return false;

}

void ProductoController::saveFile()

{

try

{

fstream archivoProductos;

archivoProductos.open("../data/productos.csv", ios::out);

if (archivoProductos.is\_open())

{

for (Producto obj : vectorProducto)

{

archivoProductos << obj.getCodProducto() << ","

<< obj.getCodMarca() << ","

<< obj.getNombre() << ","

<< obj.getPrecioUnitario() << ","

<< obj.getStock() << ","

<< obj.getNumComponentes() << ",";

for (Componente x : obj.getVectorComponentes())

archivoProductos << x.getNombre() << ","

<< x.getCantidad() << ",";

archivoProductos << endl;

}

archivoProductos.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void ProductoController::getFile()

{

try

{

int i;

int j;

size\_t posi;

string linea;

fstream archivoProductos;

vectorProducto.clear();

archivoProductos.open("../data/productos.csv", ios::in);

if (archivoProductos.is\_open())

{

while (!archivoProductos.eof() && getline(archivoProductos, linea))

{

vector<string> temporal = {};

vector<Componente> temporalComponentes = {};

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

for (int i = 6; i < temporal.size(); i += 2)

{

Componente objComponente(temporal[i], temporal[i + 1]);

temporalComponentes.push\_back(objComponente);

}

Producto obj;

obj.setCodProducto(stoi(temporal[0]));

obj.setCodMarca(stoi(temporal[1]));

obj.setNombre(temporal[2]);

obj.setPrecioUnitario(stod(temporal[3]));

obj.setStock(stoi(temporal[4]));

obj.setComponentes(temporalComponentes);

cout << "Loading Producto " << temporal[2] << " - " << obj.getNombre() << endl;

add(obj);

}

archivoProductos.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al leer el archivo";

}

}

#endif // PRODUCTOCONTROLLER\_H

#ifndef PROGRAMCONTROLLER\_H

#define PROGRAMCONTROLLER\_H

#include <iostream>

#include <string>

using namespace std;

/\*

\*/

class programController

{

private:

bool activeSesion = false;

bool isSupervisor = false;

bool isAdmin = false;

int sesionKey;

public:

programController();

~programController();

void closeSesion();

void openSesion(int, bool, bool);

bool getActiveSesion();

bool getIsSupervisor();

bool getIsAdmin();

int getSesionKey();

};

programController::programController()

{

activeSesion = false;

isSupervisor = false;

isAdmin = false;

sesionKey = -1;

}

programController::~programController() {}

void programController::closeSesion()

{

activeSesion = false;

isSupervisor = false;

isAdmin = false;

sesionKey = -1;

}

void programController::openSesion(int key, bool isSupervisor = false, bool isAdmin = false)

{

activeSesion = true;

this->isSupervisor = isSupervisor;

this->isAdmin = isAdmin;

sesionKey = key;

}

bool programController::getActiveSesion() { return activeSesion; }

bool programController::getIsSupervisor() { return isSupervisor; }

bool programController::getIsAdmin() { return isAdmin; }

int programController::getSesionKey() { return sesionKey; }

#endif

#ifndef PROVEEDORCONTROLLER\_H

#define PROVEEDORCONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/Proveedor.cpp"

class ProveedorController

{

private:

vector<Proveedor> vectorProveedor;

public:

ProveedorController();

int getCorrelativo();

void add(Proveedor);

void modify(Proveedor, int);

int size();

Proveedor get(int);

int getPosRUC(long long int);

void saveFile();

void copyFile();

};

ProveedorController::ProveedorController()

{

copyFile();

}

int ProveedorController::getCorrelativo()

{

if (size() == 0)

{

return 1;

}

else

{

return vectorProveedor[size() - 1].getCodProveedor() + 1;

}

}

void ProveedorController::add(Proveedor obj)

{

vectorProveedor.push\_back(obj);

}

void ProveedorController::modify(Proveedor temp, int obj)

{

vectorProveedor[obj] = temp;

}

int ProveedorController::size()

{

return vectorProveedor.size();

}

Proveedor ProveedorController::get(int pos)

{

return vectorProveedor[pos];

}

int ProveedorController::getPosRUC(long long int RUC)

{

for (Proveedor x : vectorProveedor)

{

if(x.getDocumento()==RUC)

return x.getCodProveedor();

}

return -1;

}

void ProveedorController::saveFile()

{

try

{

fstream archivoProveedores;

archivoProveedores.open("../data/proveedor.csv", ios::app);

if (archivoProveedores.is\_open())

{

Proveedor obj = vectorProveedor.back();

archivoProveedores << obj.getCodProveedor() << ","

<< obj.getNombre() << ","

<< obj.getTipoDocumento() << ","

<< obj.getDocumento() << ",";

archivoProveedores << endl;

archivoProveedores.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void ProveedorController::copyFile()

{

try

{

int i;

size\_t posi;

string linea;

fstream archivoProveedores;

archivoProveedores.open("../data/proveedor.csv", ios::in);

if (archivoProveedores.is\_open())

{

while (!archivoProveedores.eof() && getline(archivoProveedores, linea))

{

vector<string> temporal;

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

Proveedor obj;

obj.setCodProveedor(stoi(temporal[0]));

obj.setNombre(temporal[1]);

obj.setTipoDocumento(temporal[2]);

obj.setDocumento(stoll(temporal[3]));

add(obj);

}

archivoProveedores.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al leer el archivo";

}

}

#endif // PROVEEDORCONTROLLER\_H

#ifndef USUARIOCONTROLLER\_H

#define USUARIOCONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/Usuario.cpp"

#include "../controller/Encrypter/sha256.cpp"

class usuarioController

{

private:

vector<Usuario> vectorUsuario;

public:

usuarioController();

~usuarioController();

void add(Usuario);

int size();

Usuario get(int);

void modify(Usuario, int);

bool validarSesion(string, string); // nos valida si existe una sesiÃ³n.

bool existeUsuario(string);

bool existeUsuario(string, int&);

bool existeAdministrador();

int getNewCodUsuario();

Usuario getUsuario(int); // nos devuelve el usuario por medio de su key. (censurar la contraseÃ±a).

Usuario getUsuario(string, string); // nos devuelve el usuario por medio de su usuario y contraseÃ±a.

void archGrabarDatos();

void archRecuperarDatos();

};

usuarioController::usuarioController()

{

archRecuperarDatos();

}

usuarioController::~usuarioController() {}

void usuarioController::add(Usuario obj) { vectorUsuario.push\_back(obj); }

int usuarioController::size() { return vectorUsuario.size(); }

Usuario usuarioController::get(int pos) { return vectorUsuario[pos]; }

void usuarioController::modify(Usuario obj, int pos) { vectorUsuario[pos] = obj; }

bool usuarioController::validarSesion(string username, string contrasena)

{

int i = 0;

bool found = false;

found = existeUsuario(username,i);

cout<<i<<" "<<vectorUsuario[i - 1].getCodUsuario()<<endl;

if (found)

{

if (sha256(contrasena) == vectorUsuario[i - 1].getContrasena())

return true;

else

return false; // "Username y contraseÃ±a incorrectos y/o no registrados en nuestra base de datos."

}

else

return false; // "Username y contraseÃ±a incorrectos y/o no registrados en nuestra base de datos."

}

bool usuarioController::existeUsuario(string username)

{

int i = 0;

bool found = false;

while (i < (int)vectorUsuario.size() && !found)

{

if (aMinuscula(vectorUsuario[i].getUsername()) == aMinuscula(username))

found = true;

i++;

}

return found;

}

bool usuarioController::existeUsuario(string username,int &e)

{

int i = 0;

bool found = false;

while (i < (int)vectorUsuario.size() && !found)

{

if (aMinuscula(vectorUsuario[i].getUsername()) == aMinuscula(username))

found = true;

i++;

}

e=i;

return found;

}

bool usuarioController::existeAdministrador()

{

archRecuperarDatos();

for(Usuario x:vectorUsuario)

if(x.getTipoUsuario()=="Administrador")

return true;

return false;

}

int usuarioController::getNewCodUsuario() { return vectorUsuario.size(); }

Usuario usuarioController::getUsuario(int key)

{

int i = 0;

bool found = false;

while (i < (int)vectorUsuario.size() && !found)

{

if (vectorUsuario[i].getCodUsuario() == key)

found = true;

i++;

}

Usuario objSalida = vectorUsuario[i - 1];

return objSalida;

}

Usuario usuarioController::getUsuario(string username, string contrasena)

{

int i = 0;

bool found = false;

if (!validarSesion(username, contrasena))

{

Usuario objInvalido;

return objInvalido;

}

else

{

while (i < (int)vectorUsuario.size() && !found)

{

if (vectorUsuario[i].getUsername() == username)

found = true;

i++;

}

Usuario objSalida = vectorUsuario[i - 1];

objSalida.setContrasena("\*\*\*\*\*\*\*\*");

return objSalida;

}

}

void usuarioController::archGrabarDatos()

{

try

{

ofstream archivoUsuarios;

archivoUsuarios.open("../data/usuarios.bin", ios::out|ios::binary);

if (archivoUsuarios.is\_open())

{

vector<string> vectorLineas;

for (Usuario obj : vectorUsuario)

{

string linea = obj.getTipoDocumento() + "," +

to\_string(obj.getNumDocumento()) + "," +

obj.getUsername() + "," +

obj.getContrasena() + "," +

obj.getNombre() + "," +

obj.getApellidos() + "," +

obj.getTipoUsuario() + "," +

to\_string(obj.getCodUsuario()) + ",";

vectorLineas.push\_back(linea);

}

for (const auto& str : vectorLineas) {

int tamanoStr = str.size();

archivoUsuarios.write(reinterpret\_cast<const char\*>(&tamanoStr), sizeof(tamanoStr));

archivoUsuarios.write(str.c\_str(), tamanoStr);

}

}

archivoUsuarios.close();

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void usuarioController::archRecuperarDatos()

{

vectorUsuario.clear();

try

{

int i;

size\_t posi;

string linea;

vector<string> temporal;

ifstream archivoUsuarios;

archivoUsuarios.open("../data/usuarios.bin", ios::in|ios::binary);

if (archivoUsuarios.is\_open())

{

vector<string> vectorDeStrings;

int tamanoStr;

while (archivoUsuarios.read(reinterpret\_cast<char\*>(&tamanoStr), sizeof(tamanoStr))) {

vector<char> buffer(tamanoStr);

archivoUsuarios.read(buffer.data(), tamanoStr);

vectorDeStrings.emplace\_back(buffer.begin(), buffer.end());

}

for (string x : vectorDeStrings)

{

temporal.clear();

i = 0;

while ((posi = x.find(",")) != string::npos)

{

temporal.push\_back(x.substr(0, posi));

x.erase(0, posi + 1);

i++;

}

Usuario usuario;

usuario.setDocumento(temporal[0]);

usuario.setNumDocumento(stoi(temporal[1]));

usuario.setUsername(temporal[2]);

usuario.setContrasena(temporal[3]);

cout<<temporal[3]<<" ";

usuario.setNombre(temporal[4]);

usuario.setApellidos(temporal[5]);

usuario.setTipoUsuario(temporal[6]);

usuario.setCodUsuario(stoi(temporal[7]));

cout << "Usuario " << temporal[7] << " cargado..." << endl;

Sleep(1);

add(usuario);

}

}

archivoUsuarios.close();

}

catch(exception e)

{

cout << "Ocurrio un error al momento de copiar en el archivo";

}

}

#endif

#ifndef VENTACONTROLLER\_H

#define VENTACONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/Venta.cpp"

using namespace std;

class VentaController

{

private:

vector<Venta> vectorVenta;

public:

VentaController();

void add(Venta);

int getCorrelativo();

void modify(Venta, int);

int size();

Venta get(int);

void saveFile();

void copyFile();

};

VentaController::VentaController()

{

copyFile();

}

void VentaController::add(Venta obj)

{

vectorVenta.push\_back(obj);

}

int VentaController::getCorrelativo()

{

if (size() == 0)

{

return 1;

}

else

{

return vectorVenta[size() - 1].getCodVenta() + 1;

}

}

void VentaController::modify(Venta temp, int obj)

{

vectorVenta[obj] = temp;

}

int VentaController::size()

{

return vectorVenta.size();

}

Venta VentaController::get(int pos)

{

return vectorVenta[pos];

}

void VentaController::saveFile()

{

try

{

fstream archivoVentas;

archivoVentas.open("../data/ventas.csv", ios::app);

if (archivoVentas.is\_open())

{

Venta obj = vectorVenta.back();

archivoVentas << obj.getCodVenta() << ","

<< obj.getCodCliente() << ","

<< obj.getCodUsuario() << ","

<< obj.getFecha() << ","

<< obj.getMonto() << ","

<< obj.getEstado() << ","

<< endl;

archivoVentas.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void VentaController::copyFile()

{

try

{

int i;

size\_t posi;

string linea;

fstream archivoVentas;

archivoVentas.open("../data/ventas.csv", ios::in);

if (archivoVentas.is\_open())

{

while (!archivoVentas.eof() && getline(archivoVentas, linea))

{

vector<string> temporal;

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

Venta obj;

obj.setCodVenta(stoi(temporal[0]));

obj.setCodCliente(stoi(temporal[1]));

obj.setCodUsuario(stoi(temporal[2]));

obj.setFecha(temporal[3]);

obj.setMonto(stod(temporal[4]));

if (temporal[5] == "true")

{

obj.setEstado(true);

}

else

{

obj.setEstado(false);

}

add(obj);

}

archivoVentas.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al leer el archivo";

system("pause");

}

}

#endif // VENTACONTROLLER\_H

#ifndef VENTADCONTROLLER\_H

#define VENTADCONTROLLER\_H

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include "../model/VentaD.cpp"

using namespace std;

class VentaDController

{

private:

vector<VentaD> vectorVentaD;

public:

VentaDController();

void add(VentaD);

void modify(VentaD, int);

int size();

VentaD get(int);

void saveFile();

void copyFile();

};

VentaDController::VentaDController()

{

copyFile();

}

void VentaDController::add(VentaD obj)

{

vectorVentaD.push\_back(obj);

}

void VentaDController::modify(VentaD temp, int obj)

{

vectorVentaD[obj] = temp;

}

int VentaDController::size()

{

return vectorVentaD.size();

}

VentaD VentaDController::get(int pos)

{

return vectorVentaD[pos];

}

void VentaDController::saveFile()

{

try

{

fstream archivoVentasD;

archivoVentasD.open("../data/ventasD.csv", ios::app);

if (archivoVentasD.is\_open())

{

VentaD obj = vectorVentaD.back();

archivoVentasD << obj.getCodVenta() << ","

<< obj.getCodProducto() << ","

<< obj.getCantidad() << ","

<< obj.getPrecio() << ","

<< obj.getMonto() << ","

<< endl;

archivoVentasD.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al momento de grabar en el archivo";

}

}

void VentaDController::copyFile()

{

try

{

int i;

size\_t posi;

string linea;

fstream archivoVentasD;

archivoVentasD.open("../data/ventasD.csv", ios::in);

if (archivoVentasD.is\_open())

{

while (!archivoVentasD.eof() && getline(archivoVentasD, linea))

{

vector<string> temporal;

i = 0;

while ((posi = linea.find(",")) != string::npos)

{

temporal.push\_back(linea.substr(0, posi));

linea.erase(0, posi + 1);

i++;

}

VentaD obj;

obj.setCodVenta(stoi(temporal[0]));

obj.setCodProducto(stoi(temporal[1]));

obj.setCantidad(stoi(temporal[2]));

obj.setPrecio(stod(temporal[3]));

obj.setMonto(stod(temporal[4]));

add(obj);

}

archivoVentasD.close();

}

}

catch (exception e)

{

cout << "Ocurrio un error al leer el archivo";

system("pause");

}

}

#endif // VENTADCONTROLLER\_H

# CONCLUSIONES

En conclusión, la creación de un programa en C++ para la gestión de ventas y el inventario en la Farmacia “Miriam” se presenta como una solución eficiente y efectiva para abordar los desafíos actuales en la gestión manual de estas áreas. El programa permite automatizar las tareas de venta y gestión de inventario, reduciendo el tiempo y los errores en estas actividades, y permitiendo una mejor gestión y organización de los productos. Además, el programa cuenta con una interfaz de usuario amigable y una gestión de archivos bien organizada, lo que mejorará la experiencia del cliente y del personal de la farmacia. En última instancia, se espera que el programa ayude a mejorar la eficiencia y rentabilidad de la farmacia “Miriam”, lo que beneficiará tanto al negocio como a sus clientes.

# GLOSARIO DE TÉRMINOS

1Kárdex: Kárdex es un archivo que administra artículos en el almacén de una empresa. Registra los cambios en los bienes, materias primas o productos de la empresa. Se enfoca principalmente en el movimiento de productos comerciales hacia y desde el almacén. (Gasbarrino, 2023)

# REFERENCIAS

Gasbarrino, S. (2023, 20 enero). *Kardex: qué es y cómo hacerlo en Excel*. https://blog.hubspot.es/sales/que-es-un-kardex