

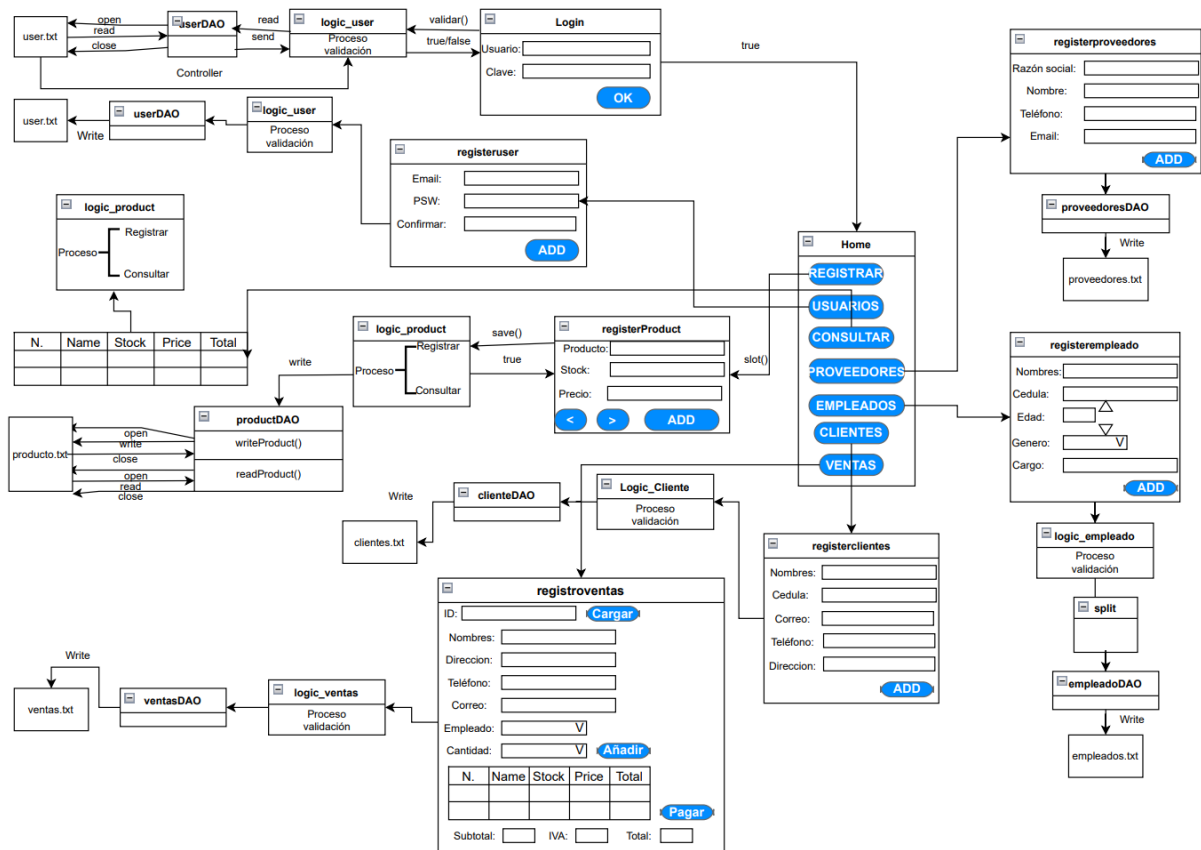
**Guía de Práctica de aplicación y experimentación de los aprendizajes de la Universidad  
Politécnica Salesiana**

<b>Carrera:</b>	<b>CIENCIAS DE LA COMPUTACIÓN</b>
<b>Nivel:</b>	<b>2do</b>
<b>Asignatura:</b>	<b>PROGRAMACIÓN ORIENTADA A OBJETOS</b>
<b>Desarrollado por:</b>	<b>Cruz David, Espinosa Marco, Pilatuña Carlos</b>
<b>Grupo:</b>	<b>1</b>
<b>Resultados de Aprendizaje:</b>	Maneja errores y guarda información en dispositivos de almacenamiento secundarios.
<b>Indicador de logro:</b>	Desarrolla aplicaciones con capacidad de persistir datos en archivos. Implementa control de excepciones en el desarrollo de aplicaciones
<b>Práctica/Deber Número:</b>	<b>Practica N°11</b>
<b>Horas Dedicadas:</b>	<b>4 horas</b>

**DESCRIPCIÓN DE LA PRÁCTICA:**

Utilizando el software que hemos desarrollado en clase, vamos a implementar los siguientes requerimientos:

1. Descargar el ejercicio desde GitHub: Enlace al repositorio
2. Registrar en el sistema: - Clientes - Proveedores - Empleados - Registrar ventas: - Actualizar el stock de productos - Vincular clientes y vendedores involucrados en la venta - Ajusta el sistema para cumplir con estos requerimientos.
3. Crear un archivo para cada proceso: clientes.txt, proveedores.txt, empleados.txt y ventas.txt y mantener la estructura siguiendo el patrón MVC



CLASE: Cliente

**Cliente.h**

```
#ifndef CLIENTE_H
#define CLIENTE_H
#include <string>
using namespace std;
```

```
class Cliente {
private:
    string nombres;
    string cedula;
    string correo;
    string telefono;
    string direccion;
```

```
public:
    Cliente();
```

```

    Cliente(string nombres, string cedula, string correo, string telefono, string direccion);
    ~Cliente();

    string getNombres() const;
    string getCedula() const;
    string getCorreo() const;
    string getTelefono() const;
    string getDireccion() const;

    void setNombres(string nombres);
    void setCedula(string cedula);
    void setCorreo(string correo);
    void setTelefono(string telefono);
    void setDireccion(string direccion);
};
#endif // CLIENTE_H

```

#### **Cliente.cpp**

```

#include "Headers/Cliente.h"
Cliente::Cliente() {
    this->nombres = "";
    this->cedula = "";
    this->correo = "";
    this->telefono = "";
    this->direccion = "";
}

Cliente::Cliente(string nombres, string cedula, string correo, string telefono, string
direccion)
    : nombres(nombres), cedula(cedula), correo(correo), telefono(telefono),
direccion(direccion) {}

Cliente::~Cliente() {}

string Cliente::getNombres() const {
    return nombres;
}

string Cliente::getCedula() const {
    return cedula;
}

string Cliente::getCorreo() const {
    return correo;
}

```

```

string Cliente::getTelefono() const {
    return telefono;
}

string Cliente::getDireccion() const {
    return direccion;
}

void Cliente::setNombres(string nombres) {
    this->nombres = nombres;
}

void Cliente::setCedula(string cedula) {
    this->cedula = cedula;
}

void Cliente::setCorreo(string correo) {
    this->correo = correo;
}

void Cliente::setTelefono(string telefono) {
    this->telefono = telefono;
}

void Cliente::setDireccion(string direccion) {
    this->direccion = direccion;
}

```

Clase: ClienteDAO

### ClienteDAO.h

```

#ifndef CLIENTEDAO_H
#define CLIENTEDAO_H
#include "Headers/Cliente.h"
#include <vector>
#include <fstream>

class ClienteDAO {
private:
    Cliente cliente;
    fstream archivo;

public:
    ClienteDAO();
    ClienteDAO(const Cliente&);
    vector<string> loadClientes();
    void writeCliente(const Cliente&);
};

#endif // CLIENTEDAO_H

```

## ClienteDAO.cpp

```
#include "Headers/ClienteDAO.h"
#include <iostream>

ClienteDAO::ClienteDAO() {}

ClienteDAO::ClienteDAO(const Cliente& c) : cliente(c) {}

vector<string> ClienteDAO::loadClientes() {
    vector<string> clientes;
    archivo.open("C://Ejercicio5//clientes.txt", ios::in);
    if (archivo.is_open()) {
        string linea = "";
        while (getline(archivo, linea)) {
            clientes.push_back(linea);
        }
        archivo.close();
    }
    return clientes;
}

void ClienteDAO::writeCliente(const Cliente& c) {
    archivo.open("C://Ejercicio5//clientes.txt", ios::app);
    if (archivo.is_open()) {
        archivo << c.getNombres() << ";" << c.getCedula() << ";" << c.getCorreo() << ";" <<
        c.getTelefono() << ";" << c.getDireccion() << endl;
        archivo.close();
    }
}
```

Clase: empleado

## empleado.h

```
#ifndef EMPLEADO_H
#define EMPLEADO_H
#include <string>

using namespace std;

class employee {
private:
```

```

        string nombres;
        long cedula;
        int edad;
        string genero;
        string cargo;
    public:
        employee();
        ~employee();
        employee(string, long, int, string, string);

        string getNombres();
        long getCedula();
        int getEdad();
        string getGenero();
        string getCargo();

        void setNombre(string);
        void setCedula(long);
        void setEdad(int);
        void setGenero(string);
        void setCargo(string);

        string information() const;
    };
#endif // EMPLEADO_H

```

#### **empleado.cpp**

```

#include "Headers/empleado.h"
employee::employee() {
    this->nombres = "";
    this->cedula = 0;
    this->edad = 0;
    this->genero = "";
    this->cargo = "";
}

employee::~employee() {}

employee::employee(string n, long ce, int e, string g, string ca)
    : nombres(n), cedula(ce), edad(e), genero(g), cargo(ca) {}

string employee::getNombres() {
    return nombres;
}

long employee::getCedula() {
    return cedula;
}

```

```

    }

    int employee::getEdad() {
        return edad;
    }

    string employee::getGenero() {
        return genero;
    }

    string employee::getCargo() {
        return cargo;
    }

    void employee::setNombre(string n) {
        this->nombres = n;
    }

    void employee::setCedula(long ce) {
        this->cedula = ce;
    }

    void employee::setEdad(int e) {
        this->edad = e;
    }

    void employee::setGenero(string g) {
        this->genero = g;
    }

    void employee::setCargo(string ca) {
        this->cargo = ca;
    }
    string employee::information() const {
        return nombres + ";" + to_string(cedula) + ";" + to_string(edad) + ";" + genero + ";" +
        cargo;
    }
}

```

Clase : empleadoDAO

**empleadoDAO.h**

```

#ifndef EMPLEADODAO_H
#define EMPLEADODAO_H
#include "Headers/empleado.h"
#include <vector>
#include <fstream>
#include <QString>

```

```

class empleadoDAO {
private:
    employee empleado;
    fstream archivo;
    string path = "C://Ejercicio5//empleados.txt";

public:
    empleadoDAO();
    empleadoDAO(const employee&);
    void writeEmployees();
    vector<string> loadEmployees(); // Leer todos los registros del archivo y almacenarlos en
    el vector
};

```

```
#endif // EMPLEADODAO_H
```

### **empleadoDAO.cpp**

```

#include "Headers/empleadoDAO.h"
#include "Headers/empleado.h"
#include <iostream>
#include <sstream>

using namespace std;

empleadoDAO::empleadoDAO() {}

empleadoDAO::empleadoDAO(const employee& e) {
    this->empleado = e;
}

vector<string> empleadoDAO::loadEmployees() {
    vector<string> empleados;
    archivo.open(path, ios::in);
    if (archivo.is_open()) {
        string linea = "";
        while (getline(archivo, linea)) {
            empleados.push_back(linea);
        }
        archivo.close();
    }
    return empleados;
}

void empleadoDAO::writeEmployees() {
    archivo.open(path, ios::app);
    if (archivo.is_open()) {
        archivo << empleado.information() << std::endl;
    }
}

```



```
        archivo.close();
    }
}
```

Clase: Logic\_Cliente

### Logic\_Cliente.h

```
#ifndef LOGIC_CLIENTE_H
#define LOGIC_CLIENTE_H

#include "Headers/ClienteDAO.h"

class Logic_Cliente {
private:
    ClienteDAO cdao;

public:
    bool validarCliente(string cedula);
    bool registrarCliente(const Cliente& cliente);
};

#endif // LOGIC_CLIENTE_H
```

### Logic\_Cliente.cpp

```
#include "Headers/logic_cliente.h"
#include "Headers/clienteDAO.h"
#include "Headers/cliente.h" // Asegúrate de incluir el archivo de cabecera de Cliente
#include <sstream>

// Función split para dividir una cadena en un vector de cadenas según un delimitador
std::vector<std::string> split(std::string_view s, char delimiter) {
    std::vector<std::string> tokens;
    std::string token;
    std::istringstream tokenStream(s.data());
    while (std::getline(tokenStream, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}

bool Logic_Cliente::validarCliente(std::string cedula) {
    std::vector<std::string> clientes = cdao.loadClientes();
    for (const std::string& cliente : clientes) {
        std::vector<std::string> datosCliente = split(cliente, ';');
        if (datosCliente.size() >= 2 && datosCliente[1] == cedula) {
            return true;
        }
    }
    return false;
}
```

```

    }
}
return false;
}

bool Logic_Cliente::registrarCliente(const Cliente& cliente) {
    std::vector<std::string> clientes = cdao.loadClientes();
    for (const std::string& clienteStr : clientes) {
        std::vector<std::string> datosCliente = split(clienteStr, ',');
        if (datosCliente.size() >= 2 && datosCliente[1] == cliente.getCedula()) {
            return false; // Cliente ya existe
        }
    }
    cdao.writeCliente(cliente);
    return true;
}

```

Clase: logic\_employee

#### logic\_employee.h

```

#ifndef LOGIC_EMPLOYEE_H
#define LOGIC_EMPLOYEE_H
#include "Headers/empleado.h"
#include "Headers/empleadoDAO.h"
class logic_employee {
private:
    empleadoDAO edao;
    employee e;
public:
    bool registrarEmpleado(string &nombres, long &cedula, int &edad, string &genero,
        string &cargo);
};

```

#endif // LOGIC\_EMPLOYEE\_H

#### logic\_employee.cpp

```

#include "Headers/logic_employee.h"
#include <string>

using namespace std;

bool logic_employee::registrarEmpleado(string &nombres, long &cedula, int &edad, string
&genero, string &cargo) {
    employee empleado(nombres, cedula, edad, genero, cargo);
    empleadoDAO edao(empleado);
    edao.writeEmployees();
}

```

```
        return true;
    }
}
```

Clase: logic\_product

#### **logic\_product.h**

```
#ifndef LOGIC_PRODUCT_H
#define LOGIC_PRODUCT_H
#include "Headers/productDAO.h"
#include <vector>
#include <string>

class logic_product {
public:
    productDAO pdao;
    logic_product();
    ~logic_product();
    bool save(std::string, int, double);
    std::vector<std::string> loadInfoProduct(int); // Permite recorrer cada producto
    almacenado
    int loadTotalProducts(); // Nuevo método para obtener el total de productos
};
```

```
#endif // LOGIC_PRODUCT_H
```

#### **logic\_product.cpp**

```
#include "Headers/logic_product.h"

logic_product::logic_product() {}

logic_product::~~logic_product() {}

bool logic_product::save(std::string nameProduct, int stock, double price) {
    product p(nameProduct, stock, price);
    productDAO pdao(p);
    pdao.writeProducts();
    return true;
}

std::vector<std::string> logic_product::loadInfoProduct(int index) {
    std::vector<std::string> infoP;
    std::vector<product> products = pdao.readProducts();
    int size = products.size();
    if (index >= 0 && index < size) {
        infoP.push_back(products[index].getName());
        infoP.push_back(std::to_string(products[index].getStock()));
        infoP.push_back(std::to_string(products[index].getPrice()));
    }
    return infoP;
}
```

```

int logic_product::loadTotalProducts() {
    std::vector<product> products = pdao.readProducts();
    return products.size();
}

```

Clase: logic\_user

**logic\_user.h**

```

#ifndef LOGIC_USER_H
#define LOGIC_USER_H
#include "Headers/userDAO.h"
class logic_user{
private:
    userDAO udao;
    user u;
public:
    bool validar(string u,string p);
};

```

#endif // LOGIC\_USER\_H

**logic\_user.cpp**

```

#include "Headers/logic_user.h"
bool logic_user::validar(string u,string p){
    bool access=false;
    vector<string> users=udao.loadUsers();//carga todos los usuarios
    for(const string & usuario:users){
        if(usuario==(u+";"+"p)){
            access=true;
        }
    }

    return access;
}

```

Clase: product

**product.h**

```

#ifndef PRODUCT_H
#define PRODUCT_H
#include <string>
using namespace std;

class product{
private:
    string name;
    int stock;
    double price;
public:
    product();
    ~product();

```

```
    product(string,int,double);  
    string getName()const;  
    int getStock()const;  
    double getPrice()const;  
    void setName(string);  
    void setStock(int);  
    void setPrice(double);  
    string information()const;  
};
```

```
#endif // PRODUCT_H
```

### **product.cpp**

```
#include "Headers/product.h"  
product::product(){  
    this->name="";  
    this->price=0;  
    this->stock=0;  
}  
product::~~product(){  
  
}  
product::product(string n,int s,double p){  
    this->name=n;  
    this->price=p;  
    this->stock=s;  
}  
string product::getName()const{  
    return name;  
}  
int product::getStock()const{  
    return stock;  
}  
double product::getPrice()const{  
    return price;  
}  
void product::setName(string n){  
    this->name=n;  
}  
void product::setStock(int s){  
    this->stock=s;  
}  
void product::setPrice(double p){  
    this->price=p;  
}  
string product::information()const{  
    return name+";"+to_string(stock)+";"+to_string(price)+"\n";  
}  
}
```

Clase: productDAO

**productDAO.h**

```
#ifndef PRODUCTDAO_H
#define PRODUCTDAO_H

#include "Headers/product.h"
#include <fstream>
#include <vector>
class productDAO{
private:
    product p;
    fstream archivo;
    string path="C://Ejercicio5//productos.txt";
public:
    productDAO();
    ~productDAO();
    productDAO(const product&);
    void writeProducts();
    vector<product> readProducts();
};
```

#endif // PRODUCTDAO\_H

**productDAO.cpp**

```
#include "Headers/productDAO.h"
#include <QDebug>
#include <string>
#include <sstream>
using namespace std;

vector<string> split(string str, char delimitador) {
    vector<string> tokens;
    stringstream ss(str);
    string item;
    while (getline(ss, item, delimitador)) {
        tokens.push_back(item);
    }
    return tokens;
}

productDAO::productDAO() {}

productDAO::~~productDAO() {}
```

```

productDAO::productDAO(const product& p_) : p(p_) {}

void productDAO::writeProducts() {
    archivo.open(path, ios::app);
    if (archivo.is_open()) {
        archivo << p.information() << endl;
        archivo.close();
    }
}

vector<product> productDAO::readProducts() {
    vector<product> products;
    vector<string> datafile; // recupera el contenido del archivo
    archivo.open(path, ios::in);
    if (archivo.is_open()) {
        string linea;
        while (getline(archivo, linea)) {
            datafile.push_back(linea);
        }
        archivo.close();
    }
    for (const auto &str : datafile) {
        vector<string> tokens = split(str, ','); // separa por ;
        if (tokens.size() == 3) {
            product p;
            p.setName(tokens[0]);
            p.setStock(stoi(tokens[1]));
            p.setPrice(stod(tokens[2]));
            products.push_back(p);
        }
    }
    return products;
}

```

Clase: proveedores

#### **proveedores.h**

```

#ifndef PROVEEDORES_H
#define PROVEEDORES_H
#include<string>
using namespace std;
class proveedores{
private:
    string razonSocial;
    string nombres;
    string telefono;
    string email;
public:
    proveedores();
    ~proveedores();

```

```
proveedores(string,string,string,string);
string getRazonSocial()const;
string getNombres()const;
string getTelefono()const;
string getEmail()const;
void setRazonSocial(string);
void setNombres(string);
void setTelefono(string);
void setEmail(string);
};
```

```
#endif // PROVEEDORES_H
```

#### **proveedores.cpp**

```
#include "Headers/proveedores.h"
proveedores::proveedores(){
    this->razonSocial="";
    this->telefono="";
    this->nombres="";
    this->email="";
}
proveedores::~proveedores(){

}
proveedores::proveedores(string r,string t,string n ,string
e):razonSocial(r),nombres(n),telefono(t),email(e){

}
string proveedores::getRazonSocial()const{
    return razonSocial;
}
string proveedores::getNombres()const{
    return nombres;
}
string proveedores::getTelefono()const{
    return telefono;
}
string proveedores::getEmail()const{
    return email;
}
void proveedores::setRazonSocial(string r){
    razonSocial=r;
}
void proveedores::setNombres(string n){
    nombres=n;
}
void proveedores::setTelefono(string t){
    telefono=t;
}
```



```
void proveedores::setEmail(string e){  
    email=e;  
}
```

Clase: proveedoresDAO

#### **proveedoresDAO.h**

```
#ifndef PROVEEDORESDAO_H  
#define PROVEEDORESDAO_H  
#include "Headers/proveedores.h"  
#include <vector>  
#include <fstream>  
class proveedoresDAO{  
private:  
    proveedores pro;  
    fstream archivo;  
private:  
    proveedoresDAO();  
    proveedoresDAO(const proveedores& p);  
    vector<string> loadProveedores();  
};
```

```
#endif // PROVEEDORESDAO_H
```

#### **proveedoresDAO.cpp**

```
#include "Headers/proveedoresDAO.h"  
#include <iostream>  
proveedoresDAO::proveedoresDAO(){  
  
}  
proveedoresDAO::proveedoresDAO(const proveedores& p){  
    this->pro=p;  
}  
vector<string> proveedoresDAO::loadProveedores(){  
    vector<string> proveedores;  
    archivo.open("C://Ejercicio5//proveedores.txt",ios::in);  
    if(archivo.is_open()){  
        string linea="";  
        while(getline(archivo,linea)){  
            cout<<linea<<endl;  
            proveedores.push_back(linea);  
        }  
        archivo.close();  
    }  
    return proveedores;  
}
```

Clase: user

#### **user.h**

```
#ifndef USER_H  
#define USER_H  
#include <string>
```

```
using namespace std;
class user{
private:
    string nameUser, psw;
public:
    user();
    ~user();
    user(string u ,string p);
    string getUser()const;
    string getPSW()const;
    void setUser(string u);
    void setPSW(string p);
};
```

```
#endif // USER_H
```

#### **user.cpp**

```
#include "Headers/user.h"

user::user(){
    this->nameUser="";
    this->psw="";
}
user::~user(){}
user::user(string u,string p):nameUser(u),psw(p){}
string user::getUser()const{
    return nameUser;
}
string user::getPSW()const{
    return psw;
}
void user::setUser(string u){
    nameUser=u;
}
void user::setPSW(string p){
    psw=p;
}
}
```

Clase: userDAO

#### **userDAO.h**

```
#ifndef USERDAO_H
#define USERDAO_H
#include "Headers/user.h"
#include <vector>
#include <fstream>
class userDAO{
private:
    user usuario;
    fstream archivo;
public:
```

```

        userDAO();
        userDAO(const user& u);
        vector<string> loadUsers();//leer todos los registros del archivo y almacenarlos en el
        vector
    };

#endif // USERDAO_H
userDAO.cpp
#include "Headers/userDAO.h"
#include <iostream>
userDAO::userDAO(){

}
userDAO::userDAO(const user& u){
    this->usuario=u;
}
vector<string> userDAO::loadUsers(){
    vector<string> users;
    archivo.open("C://Ejercicio5//users.txt",ios::in);
    if(archivo.is_open()){
        string linea="";
        while(getline(archivo,linea)){
            cout<<linea<<endl;
            users.push_back(linea);
        }
        archivo.close();
    }
    return users;
}

```

Clase: Home

### Home.h

```

#ifndef HOME_H
#define HOME_H
#include "Vista/registerempleado.h"
#include "Vista/registeruser.h"
#include "Vista/registerproduct.h"
#include "Vista/viewproducts.h"
#include "Vista/registerproveedores.h"
#include "Vista/registrarcientes.h"
#include "Vista/registroventas.h"
#include <QWidget>

namespace Ui {
class Home;
}

class Home : public QWidget

```

```

{
    Q_OBJECT

public:
    explicit Home(QWidget *parent = nullptr);
    ~Home();

private:
    Ui::Home *ui;
    registerProduct *uiRp;
    registeruser *uiRu;
    viewproducts *uiVp;
    registerproveedores *uiRv;
    registerEmpleado *uiRe;
    RegistrarClientes *uiRc;
    registroVentas *uiRegistroVentas;
private slots:
    void callRegisterProduct();
    void callRegisterUser();
    void viewProducts();
    void callRegisterProveedores();
    void callRegisterEmpleados();
    void callRegisterCLientes();
    void callRegistroVentas();
};

#endif // HOME_H
Home.cpp
#include "home.h"
#include "ui_home.h"

Home::Home(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Home)
    , uiRp(new registerProduct)
    , uiRu(new registeruser)
    , uiVp(new viewproducts)
    , uiRv(new registerproveedores)
    , uiRe(new registerEmpleado)
    , uiRc (new RegistrarClientes)
    , uiRegistroVentas(new registroVentas)

{
    ui->setupUi(this);
    connect(ui->btn_registrar, SIGNAL(clicked()),this,SLOT(callRegisterProduct()));
    connect(ui->btn_usuarios,SIGNAL(clicked()),this,SLOT(callRegisterUser()));
    connect(ui->btn_consultar,SIGNAL(clicked()),this,SLOT(viewProducts()));
    connect(ui->btn_proveedores,SIGNAL(clicked()),this,SLOT(callRegisterProveedores()));

```

```

connect(ui->btn_empleados,SIGNAL(clicked()),this,SLOT(callRegisterEmpleados()));
connect(ui->btn_clientes,SIGNAL(clicked()),this,SLOT(callRegisterClientes()));
connect(ui->btn_ventas, SIGNAL(clicked()), this, SLOT(callRegistroVentas()));

}

Home::~~Home()
{
    delete ui;
}
void Home::callRegisterProduct(){
    uiRp->show();
}
void Home::callRegisterUser(){
    uiRu->show();
}
void Home::viewProducts(){
    uiVp->show();
}
void Home::callRegisterProveedores(){
    uiRv->show();
}
void Home::callRegisterEmpleados(){
    uiRe->show();
}
void Home::callRegisterClientes(){
    uiRc->show();
}
void Home::callRegistroVentas()
{
    uiRegistroVentas->show();
    uiRegistroVentas->recargarDatos();
}

```

Clase: registerEmpleado

#### **registerEmpleado.h**

```

#ifndef REGISTEREMPLEADO_H
#define REGISTEREMPLEADO_H

#include <QWidget>
#include "Headers/logic_employee.h"
namespace Ui {
class registerEmpleado;
}

class registerEmpleado : public QWidget
{
    Q_OBJECT

```

```
public:
    explicit registerEmpleado(QWidget *parent = nullptr);
    ~registerEmpleado();
```

```
private:
    Ui::registerEmpleado *ui;
    logic_employee le,
    *ptrLe=nullptr;
private slots:
    void registrar();
};
```

```
#endif // REGISTEREMPLEADO_H
```

#### **registerEmpleado.cpp**

```
#include "registerempleado.h"
#include "ui_registerempleado.h"
#include<QMessageBox>
registerEmpleado::registerEmpleado(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::registerEmpleado)
{
    ui->setupUi(this);
    ptrLe=&le;

    connect(ui->btn_addEmployee,SIGNAL(clicked()),this,SLOT(registrar()));

    ui->comboBox->addItem("Hombre");
    ui->comboBox->addItem("Mujer");

}

registerEmpleado::~registerEmpleado()
{
    delete ui;
}
```

```
void registerEmpleado::registrar(){
    string nombres = ui->txt_nombres->text().toString();
    long cedula = ui->txt_cedula->text().toLong();
    int edad = ui->spinBox->value();
    string genero = ui->comboBox->currentText().toString();
    string cargo = ui->txt_cargo->text().toString();

    logic_employee le;
    if (ptrLe->registrarEmpleado(nombres, cedula, edad, genero, cargo)){
```

```

        QMessageBox::information(this, "Registro de Empleado", "Empleado registrado con
        éxito.");
    } else {
        QMessageBox::warning(this, "Registro de Empleado", "Error al registrar el
        empleado.");
    }
}

```

Clase: registerProduct

#### registerProduct.h

```

#ifndef REGISTERPRODUCT_H
#define REGISTERPRODUCT_H

#include "Headers/logic_product.h"
#include <QWidget>
#include <vector>
#include <string>

namespace Ui {
class registerProduct;
}

class registerProduct : public QWidget
{
    Q_OBJECT

public:
    explicit registerProduct(QWidget *parent = nullptr);
    ~registerProduct();

private:
    Ui::registerProduct *ui;
    logic_product lp;
    logic_product *ptrLp;
    void clearFields();
    void loadFields(int index); // cargar los campos del producto
    int index;
    int totalProducts;

private slots:
    void saveProduct();
    void prev();
    void next();
    void newProduct();
};

```

```

#endif // REGISTERPRODUCT_H
registerProduct.cpp

```

```

#include "registerProduct.h"
#include <QMessageBox>
#include "ui_registerproduct.h"

registerProduct::registerProduct(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::registerProduct)
    , index(0)
    , totalProducts(0)
{
    ui->setupUi(this);
    ui->btn_agregar->setEnabled(false);
    ptrLp = &lp;

    connect(ui->btn_agregar, SIGNAL(clicked()), this, SLOT(saveProduct()));
    connect(ui->btn_next, SIGNAL(clicked()), this, SLOT(next()));
    connect(ui->btn_previous, SIGNAL(clicked()), this, SLOT(prev()));
    connect(ui->btn_new, SIGNAL(clicked()), this, SLOT(newProduct()));

    // Cargar el primer producto al iniciar la interfaz
    totalProducts = ptrLp->loadTotalProducts();
    if (totalProducts > 0) {
        loadFields(index);
    } else {
        QMessageBox::information(this, "Registro de Productos", "No hay productos
registrados.");
    }
}

registerProduct::~registerProduct()
{
    delete ui;
}

void registerProduct::saveProduct() {
    std::string nameProduct = ui->txt_producto->text().toStdString();
    int stock = ui->txt_unidades->text().toInt();
    double price = ui->txt_precio->text().toDouble();
    if (ptrLp->save(nameProduct, stock, price)) {
        QMessageBox::information(this, "Registro de productos", "Registro exitoso");
        clearFields();
        totalProducts++;
        index = totalProducts - 1;
    }
}

void registerProduct::clearFields() {
    ui->txt_producto->clear();

```



```

        ui->txt_precio->clear();
        ui->txt_unidades->clear();
    }

    void registerProduct::loadFields(int index) {
        ui->btn_agregar->setEnabled(false);
        std::vector<std::string> p = ptrLp->loadInfoProduct(index);
        if (p.size() > 0) {
            ui->txt_producto->setText(QString::fromStdString(p[0]));
            ui->txt_unidades->setText(QString::fromStdString(p[1]));
            ui->txt_precio->setText(QString::fromStdString(p[2]));
        } else {
            QMessageBox::information(this, "Registro de Productos", "No existen más
productos.");
        }
    }

    void registerProduct::prev() {
        if (index > 0) {
            index--;
            loadFields(index);
        } else {
            QMessageBox::information(this, "Registro de Productos", "Este es el primer
producto.");
        }
    }

    void registerProduct::next() {
        if (index < totalProducts - 1) {
            index++;
            loadFields(index);
        } else {
            QMessageBox::information(this, "Registro de Productos", "Este es el último
producto.");
        }
    }

    void registerProduct::newProduct() {
        clearFields();
        ui->btn_agregar->setEnabled(true);
    }

```

Clase: registerproveedores

#### Registerproveedores.h

```

#ifndef REGISTERPROVEEDORES_H
#define REGISTERPROVEEDORES_H

#include <QWidget>

```

```

namespace Ui {
class registerproveedores;
}

class registerproveedores : public QWidget
{
    Q_OBJECT

public:
    explicit registerproveedores(QWidget *parent = nullptr);
    ~registerproveedores();
private slots:
    void addPorveedores();
private:
    Ui::registerproveedores *ui;
    void saveUserToFile(const QString &razonSocial, const QString &nombres, const QString
&telefono,const QString &email);
};

```

```
#endif // REGISTERPROVEEDORES_H
```

#### **Registerproveedores.cpp**

```

#include "registerproveedores.h"
#include "ui_registerproveedores.h"
#include<QMessageBox>
#include<QFile>
registerproveedores::registerproveedores(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::registerproveedores)
{
    ui->setupUi(this);
    connect(ui->btn_add2, &QPushButton::clicked, this,
&registerproveedores::addPorveedores);
}

registerproveedores::~registerproveedores()
{
    delete ui;
}

void registerproveedores::addPorveedores(){
    QString razonSocial = ui->txt_RazonSocial->text();
    QString nombres = ui->txt_nombres->text();
    QString telefono = ui->txt_telefono->text();
    QString email = ui->txt_email->text();

    if (razonSocial.isEmpty() || nombres.isEmpty() || telefono.isEmpty() || email.isEmpty()) {
        QMessageBox::warning(this, "Registro de Proveedor", "Todos los campos son
obligatorios.");
        return;
    }
}

```

```

    }
    saveUserToFile(razonSocial,nombres,telefono,email);
}
void registerproveedores::saveUserToFile(const QString &razonSocial, const QString
&nombres, const QString &telefono,const QString &email){
    QFile file("C://Ejercicio5//proveedores.txt");
    if (!file.open(QIODevice::Append | QIODevice::Text)) {
        QMessageBox::critical(this, "Registro de Proveedor", "No se pudo abrir el archivo para
escribir.");
        return;
    }

    QTextStream out(&file);
    out << razonSocial << ";" << nombres << ";" << telefono << ";" << email << "\n";
    file.close();

    QMessageBox::information(this, "Registro de Proveedor", "Proveedor registrado
exitosamente.");
    ui->txt_RazonSocial->clear();
    ui->txt_nombres->clear();
    ui->txt_telefono->clear();
    ui->txt_email->clear();
}

```

Clase: registeruser

#### Registeruser.h

```

#ifndef REGISTERUSER_H
#define REGISTERUSER_H

#include <QWidget>

namespace Ui {
class registeruser;
}

class registeruser : public QWidget
{
    Q_OBJECT

public:
    explicit registeruser(QWidget *parent = nullptr);
    ~registeruser();
private slots:
    void addUser();
private:
    Ui::registeruser *ui;
    void saveUserToFile(const QString &username, const QString &password);
};

```

```
#endif // REGISTERUSER_H
```

### **Registeruser.cpp**

```
#include "registeruser.h"
#include "ui_registeruser.h"
#include <QFile>
#include <QTextStream>
#include <QMessageBox>
registeruser::registeruser(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::registeruser)
{
    ui->setupUi(this);
    connect(ui->btn_add, &QPushButton::clicked, this, &registeruser::addUser);
}

registeruser::~registeruser()
{
    delete ui;
}

void registeruser::addUser()
{
    QString username = ui->txt_user->text();
    QString password = ui->txt_psw->text();
    QString confirmPassword = ui->txt_confirmarpsw->text();

    if (username.isEmpty() || password.isEmpty() || confirmPassword.isEmpty()) {
        QMessageBox::warning(this, "Registro de Usuario", "Todos los campos son obligatorios.");
        return;
    }

    if (password != confirmPassword) {
        QMessageBox::warning(this, "Registro de Usuario", "Las contraseñas no coinciden.");
        return;
    }

    saveUserToFile(username, password);
}

void registeruser::saveUserToFile(const QString &username, const QString &password)
{
    QFile file("C://Ejercicio5//users.txt");
    if (!file.open(QIODevice::Append | QIODevice::Text)) {
        QMessageBox::critical(this, "Registro de Usuario", "No se pudo abrir el archivo para escribir.");
        return;
    }
}
```

```

QTextStream out(&file);
out << username << "," << password << "\n";
file.close();

```

```

QMessageBox::information(this, "Registro de Usuario", "Usuario registrado
exitosamente.");
ui->txt_user->clear();
ui->txt_psw->clear();
ui->txt_confirmarpsw->clear();
}

```

Clase: RegistrarClientes

#### RegistrarClientes.h

```

#ifndef REGISTRARCLIENTES_H
#define REGISTRARCLIENTES_H

#include <QWidget>
#include "Headers/Cliente.h"
#include "Headers/Logic_Cliente.h"
namespace Ui {
class RegistrarClientes;
}

class RegistrarClientes : public QWidget
{
    Q_OBJECT

public:
    explicit RegistrarClientes(QWidget *parent = nullptr);
    ~RegistrarClientes();
private slots:
    void on_addButton_clicked();
private:
    Ui::RegistrarClientes *ui;
};

```

#endif // REGISTRARCLIENTES\_H

#### RegistrarClientes.cpp

```

#include "registrarclientes.h"
#include "ui_registrarclientes.h"
#include <QMessageBox>

RegistrarClientes::RegistrarClientes(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::RegistrarClientes)
{
    ui->setupUi(this);
    connect(ui->btn_addCliente,SIGNAL(clicked()),this,SLOT(on_addButton_clicked()));
}

```

```

RegistrarClientes::~RegistrarClientes()
{
    delete ui;
}

void RegistrarClientes::on_addButton_clicked() {
    QString nombres = ui->nombresLineEdit->text();
    QString cedula = ui->cedulaLineEdit->text();
    QString correo = ui->correoLineEdit->text();
    QString telefono = ui->telefonoLineEdit->text();
    QString direccion = ui->direccionLineEdit->text();

    Cliente nuevoCliente(nombres.toStdString(), cedula.toStdString(), correo.toStdString(),
telefono.toStdString(), direccion.toStdString());

    Logic_Cliente logicCliente;
    if (logicCliente.registrarCliente(nuevoCliente)) {
        QMessageBox::information(this, "Éxito", "Cliente registrado correctamente");
    } else {
        QMessageBox::warning(this, "Error", "El cliente ya existe");
    }
}

```

Clase: registroVentas

#### registroVentas.h

```

#ifndef REGISTROVENTAS_H
#define REGISTROVENTAS_H

#include <QWidget>
#include <QString>
#include <QVector>
#include <QMap>

namespace Ui {
class registroVentas;
}

class registroVentas : public QWidget
{
    Q_OBJECT

public:
    explicit registroVentas(QWidget *parent = nullptr);
    ~registroVentas();
    void recargarDatos(); // Función para recargar datos

private slots:
    void aniadirproducto();

```

```

void pagar();
void on_lineEdit_CedulaCliente_editingFinished();

private:
    Ui::registroVentas *ui;
    QMap<QString, QStringList> clientes;
    QVector<QString> empleados;
    QMap<QString, double> productos;
    QMap<QString, int> stockProductos; // Mapa para almacenar el stock de productos

    const QString dataDirectory = "C://Ejercicio5/";

    void cargarDatosClientes();
    void cargarDatosEmpleados();
    void cargarDatosProductos();
    void actualizarTotal();
    void actualizarStock(const QString &producto, int cantidad);
};

#endif // REGISTROVENTAS_H

```

#### **registroVentas.cpp**

```

#include "registroventas.h"
#include "ui_registroventas.h"
#include <QFile>
#include <QTextStream>
#include <QMessageBox>

registroVentas::registroVentas(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::registroVentas)
{
    ui->setupUi(this);

    cargarDatosClientes();
    cargarDatosEmpleados();
    cargarDatosProductos();

    connect(ui->btn_loadCliente, &QPushButton::clicked, this,
        &registroVentas::on_lineEdit_CedulaCliente_editingFinished);
    connect(ui->btn_addProducto, &QPushButton::clicked, this,
        &registroVentas::aniadirproducto);
    connect(ui->btn_Pagar, &QPushButton::clicked, this, &registroVentas::pagar);
}

registroVentas::~registroVentas()

```

```

{
    delete ui;
}

void registroVentas::cargarDatosClientes()
{
    QFile file(dataDirectory + "clientes.txt");
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::critical(this, "Error", "No se pudo abrir el archivo de clientes.");
        return;
    }

    QTextStream in(&file);
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList fields = line.split(';'); // Cambiar la separación a punto y coma
        if (fields.size() >= 5) {
            clientes[fields[1]] = fields;
        }
    }
    file.close();
}

void registroVentas::cargarDatosEmpleados()
{
    QFile file(dataDirectory + "empleados.txt");
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::critical(this, "Error", "No se pudo abrir el archivo de empleados.");
        return;
    }

    QTextStream in(&file);
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList fields = line.split(';'); // Dividir la línea por punto y coma
        if (!fields.isEmpty()) {
            QString nombreEmpleado = fields[0]; // Usar solo el primer índice
            empleados.append(nombreEmpleado);
            ui->comboBox_Empleados->addItem(nombreEmpleado);
        }
    }
    file.close();
}

void registroVentas::cargarDatosProductos()
{
    QFile file(dataDirectory + "productos.txt");

```



```

        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QMessageBox::critical(this, "Error", "No se pudo abrir el archivo de productos.");
            return;
        }

        QTextStream in(&file);
        while (!in.atEnd()) {
            QString line = in.readLine();
            QStringList fields = line.split(';'); // Asegúrate de usar punto y coma (;) como separador
            if (fields.size() >= 3) {
                QString nombreProducto = fields[0];
                int stock = fields[1].toInt();
                double precio = fields[2].toDouble();
                productos[nombreProducto] = precio;
                stockProductos[nombreProducto] = stock; // Guardamos el stock del producto en
stockProductos
                ui->comboBox_Productos->addItem(nombreProducto); // Añadir el nombre del
producto al ComboBox
            }
        }
        file.close();
    }

    void registroVentas::on_lineEdit_CedulaCliente_editingFinished()
    {
        QString cedula = ui->lineEdit_CedulaCliente->text();
        if (clientes.contains(cedula)) {
            QStringList datos = clientes[cedula];
            ui->txt_NombreCliente->setText(datos[0]); // Nombre del cliente
            ui->txt_DireccionCliente->setText(datos[4]); // Dirección del cliente
            ui->txt_TelfCliente->setText(datos[3]); // Teléfono del cliente
            ui->txt_CorreoCliente->setText(datos[2]); // Correo del cliente
        } else {
            QMessageBox::warning(this, "Advertencia", "Cliente no encontrado.");
            ui->txt_NombreCliente->clear(); // Limpiar campos si el cliente no se encuentra
            ui->txt_DireccionCliente->clear();
            ui->txt_TelfCliente->clear();
            ui->txt_CorreoCliente->clear();
        }
    }

    void registroVentas::aniadirproducto()
    {
        QString producto = ui->comboBox_Productos->currentText();
        int cantidad = ui->lineEdit_Cantidad->text().toInt();

        if (cantidad <= 0) {
            QMessageBox::warning(this, "Advertencia", "La cantidad debe ser mayor que cero.");

```

```

        return;
    }

    // Verificar si hay suficiente stock
    if (stockProductos.contains(producto) && stockProductos[producto] >= cantidad) {
        double precio = productos[producto];
        double total = cantidad * precio;

        int row = ui->tableWidget_Compras->rowCount();
        ui->tableWidget_Compras->insertRow(row);
        ui->tableWidget_Compras->setItem(row, 0, new QTableWidgetItem(producto));
        ui->tableWidget_Compras->setItem(row, 1, new
        QTableWidgetItem(QString::number(cantidad)));
        ui->tableWidget_Compras->setItem(row, 2, new
        QTableWidgetItem(QString::number(precio)));
        ui->tableWidget_Compras->setItem(row, 3, new
        QTableWidgetItem(QString::number(total)));

        actualizarTotal();
        actualizarStock(producto, cantidad); // Actualizar el stock del producto
    } else {
        QMessageBox::warning(this, "Advertencia", "Stock insuficiente para este producto.");
    }
}

void registroVentas::actualizarTotal()
{
    double subtotal = 0;
    for (int i = 0; i < ui->tableWidget_Compras->rowCount(); ++i) {
        subtotal += ui->tableWidget_Compras->item(i, 3)->text().toDouble();
    }

    double iva = subtotal * 0.15;
    double total = subtotal + iva;

    ui->txt_Subtotal->setText(QString::number(subtotal));
    ui->txt_Iva->setText(QString::number(iva));
    ui->txt_Total->setText(QString::number(total));
}

void registroVentas::pagar()
{
    QString cedula = ui->lineEdit_CedulaCliente->text();
    QString nombreCliente = ui->txt_NombreCliente->text();
    QString direccionCliente = ui->txt_DireccionCliente->text();
    QString telfCliente = ui->txt_TelfCliente->text();
    QString correoCliente = ui->txt_CorreoCliente->text();
    QString empleado = ui->comboBox_Empleados->currentText();

```

```

QFile file(dataDirectory + "ventas.txt");
if (!file.open(QIODevice::Append | QIODevice::Text)) {
    QMessageBox::critical(this, "Error", "No se pudo abrir el archivo de ventas.");
    return;
}

QTextStream out(&file);
out << cedula << "," << nombreCliente << "," << direccionCliente << "," << telfCliente <<
", " << correoCliente << "," << empleado << "\n";

for (int i = 0; i < ui->tableWidget_Compras->rowCount(); ++i) {
    QString producto = ui->tableWidget_Compras->item(i, 0)->text();
    QString cantidad = ui->tableWidget_Compras->item(i, 1)->text();
    QString precio = ui->tableWidget_Compras->item(i, 2)->text();
    QString total = ui->tableWidget_Compras->item(i, 3)->text();

    out << producto << "," << cantidad << "," << precio << "," << total << "\n";
}

file.close();
QMessageBox::information(this, "Éxito", "Venta registrada exitosamente.");
}

void registroVentas::actualizarStock(const QString &producto, int cantidad)
{
    if (stockProductos.contains(producto)) {
        stockProductos[producto] -= cantidad; // Actualizar el stock del producto
    }
}

void registroVentas::recargarDatos()
{
    // Limpiar datos existentes
    clientes.clear();
    productos.clear();
    stockProductos.clear();
    empleados.clear();
    ui->comboBox_Productos->clear();
    ui->comboBox_Empleados->clear();

    // Recargar datos de clientes
    QFile fileClientes(dataDirectory + "clientes.txt");
    if (!fileClientes.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::critical(this, "Error", "No se pudo abrir el archivo de clientes.");
        return;
    }
}

```

```

QTextStream inClientes(&fileClientes);
while (!inClientes.atEnd()) {
    QString line = inClientes.readLine();
    QStringList fields = line.split(';'); // Cambiar la separación a punto y coma
    if (fields.size() >= 5) {
        clientes[fields[1]] = fields;
    }
}
fileClientes.close();

// Recargar datos de productos
QFile fileProductos(dataDirectory + "productos.txt");
if (!fileProductos.open(QIODevice::ReadOnly | QIODevice::Text)) {
    QMessageBox::critical(this, "Error", "No se pudo abrir el archivo de productos.");
    return;
}

QTextStream inProductos(&fileProductos);
while (!inProductos.atEnd()) {
    QString line = inProductos.readLine();
    QStringList fields = line.split(';'); // Asegúrate de usar punto y coma (;) como separador
    if (fields.size() >= 3) {
        QString nombreProducto = fields[0];
        int stock = fields[1].toInt();
        double precio = fields[2].toDouble();
        productos[nombreProducto] = precio;
        stockProductos[nombreProducto] = stock; // Guardamos el stock del producto en
stockProductos
        ui->comboBox_Productos->addItem(nombreProducto); // Añadir el nombre del
producto al ComboBox
    }
}
fileProductos.close();

// Recargar datos de empleados
QFile fileEmpleados(dataDirectory + "empleados.txt");
if (!fileEmpleados.open(QIODevice::ReadOnly | QIODevice::Text)) {
    QMessageBox::critical(this, "Error", "No se pudo abrir el archivo de empleados.");
    return;
}

QTextStream inEmpleados(&fileEmpleados);
while (!inEmpleados.atEnd()) {
    QString line = inEmpleados.readLine();
    QStringList fields = line.split(';'); // Dividir la línea por punto y coma
    if (!fields.isEmpty()) {
        QString nombreEmpleado = fields[0]; // Usar solo el primer índice
        empleados.append(nombreEmpleado);
    }
}

```

```

        ui->comboBox_Empleados->addItem(nombreEmpleado);
    }
}
fileEmpleados.close();
}

```

Clase: viewproducts

#### **Viewproducts.h**

```

#ifndef VIEWPRODUCTS_H
#define VIEWPRODUCTS_H

#include <QWidget>
#include <QTableWidget>
#include "Headers/logic_product.h"
namespace Ui {
class viewproducts;
}

class viewproducts : public QWidget
{
    Q_OBJECT

public:
    explicit viewproducts(QWidget *parent = nullptr);
    ~viewproducts();
    void loadProducts();

private:
    Ui::viewproducts *ui;
    logic_product lp;
    logic_product *ptrLp;
};

```

#endif // VIEWPRODUCTS\_H

#### **Viewproducts.cpp**

```

#include "viewproducts.h"
#include "ui_viewproducts.h"
#include <vector>
#include <string>

viewproducts::viewproducts(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::viewproducts)
    , ptrLp(&lp)
{
    ui->setupUi(this);
    loadProducts();
}

```

```

viewproducts::~~viewproducts()
{
    delete ui;
}
void viewproducts::loadProducts()
{
    std::vector<product> products = ptrLp->pdao.readProducts(); // Asumiendo que
readProducts es un método público de pdao
    ui->table_view->setRowCount(products.size());
    ui->table_view->setColumnCount(4); // Asumiendo 4 columnas: Name, Stock, Price, Total

    QStringList headers = {"Name", "Stock", "Price", "Total"};
    ui->table_view->setHorizontalHeaderLabels(headers);

    for (int i = 0; i < products.size(); ++i) {
        ui->table_view->setItem(i, 0, new
QTableWidgetItem(QString::fromStdString(products[i].getName())));
        ui->table_view->setItem(i, 1, new
QTableWidgetItem(QString::number(products[i].getStock())));
        ui->table_view->setItem(i, 2, new
QTableWidgetItem(QString::number(products[i].getPrice())));

        double total = products[i].getStock() * products[i].getPrice();
        ui->table_view->setItem(i, 3, new QTableWidgetItem(QString::number(total)));
    }
}

```

Clase: widget

#### Widget.h

```

#ifndef WIDGET_H
#define WIDGET_H

#include "Headers/logic_user.h"
#include "Vista/home.h"

#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui {
class Widget;
}
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);

```

```

        ~Widget();

private:
    Ui::Widget *ui;
    Home *uiHome;
    logic_user lu,
        *ptrLU=nullptr;
private slots:
    void validar();

};
#endif // WIDGET_H
Widget.cpp
#include "widget.h"
#include "ui_widget.h"
#include <QMessageBox>
Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
    , uiHome(new Home)
{
    ui->setupUi(this);
    ptrLU=&lu;
    connect(ui->btn_login,SIGNAL(clicked()),this,SLOT(validar()));
}

Widget::~Widget()
{
    delete ui;
}

void Widget::validar(){
    string usuario=ui->txt_user->text().toStdString();
    string psw=ui->txt_psw->text().toStdString();
    if(ptrLU->validar(usuario,psw)){
        QMessageBox::information(this,"GESTOR DE INVENTARIO","Bienvenido "+
QString::fromStdString(usuario));
        uiHome->show();
        this->hide();
    }else{
        QMessageBox::information(this,"GESTOR DE INVENTARIO","Credenciales incorrectas
"+ QString::fromStdString(usuario));
    }
}
}

```

1.2 Generar una clase main que cumpla con los siguientes requisitos planteados en el problema:

Clase: main

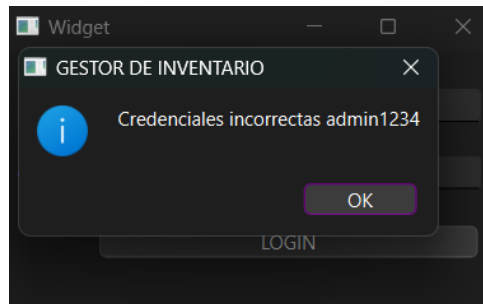
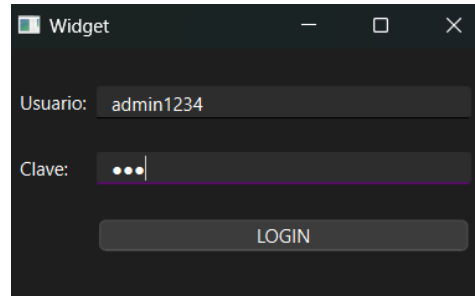
Capturas de Pantalla con cada opción ejecutada

**Main**

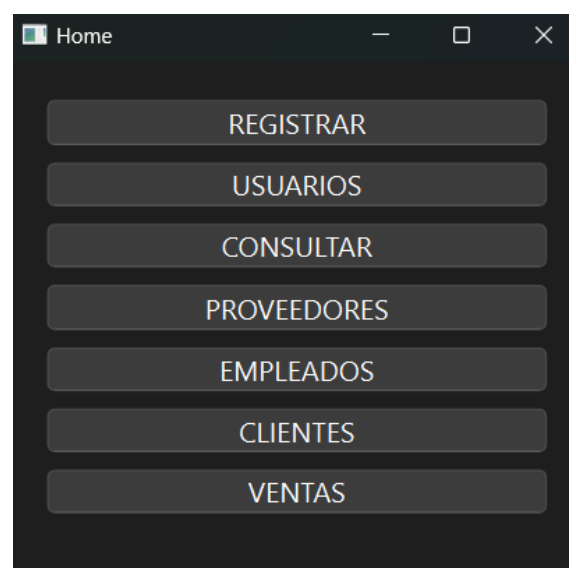
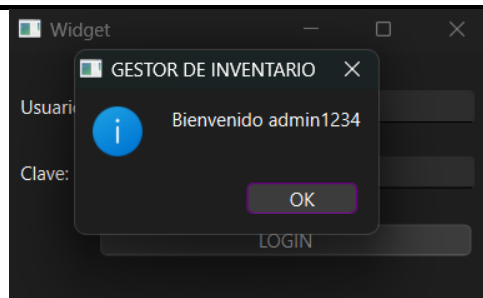
```
#include
"widget.h"

#include
<QApplication>

int main(int
argc, char
*argv[])
{
    QApplication
a(argc, argv);
    Widget w;
    w.show();
    return
a.exec();
}
```







REGISTRO DE PRODUCTOS

<< >>

Producto: monitor

Unidades: 50

Precio: 120.000000

AGREGAR

NUEVO

REGISTRO DE PRODUCTOS

<< >>

Producto: monitor

Unidades: 50

Precio: 120.000000

AGREGAR

NUEVO

Registro de produc... X

Registro exitoso

OK

REGISTRO DE USUARIOS

Usuario: Jimena Ortiz

Clave: .....

Confirmar: .....

ADD

REGISTRO DE USUARIOS

Usuario:

Clave:

Confirmar:

ADD

Registro de Usuario

Usuario registrado exitosamente.

OK

users.txt

File Edit View

```
admin1234;123
Jimena Ortiz;12345
```

LISTA DE PRODUCTOS

	Name	Stock	Price	Total
1	monitor	50	120	6000

Form

Razon social:

Nombre:

Telefono:

Email:

ADD

```
users.txt x proveedores.txt x + - □ ×
File Edit View
ElectroProveedores S.A. de C.V.;Juan Pérez García;55 1234
5678;juan.perez@electroproveedores.com
```

Form

Nombres

Cedula

Edad  ^ v

Genero  v

Cargo

ADD

Form

Nombres

Genero  v

Cargo

ADD

Registro de Empleado

Empleado registrado con éxito.

OK

```
users.txt x proveedores.tx emplea x + - □ ×
File Edit View
Julian Torres;1751578956;25;Hombre;vendedor
Andres Ruiz;1751578456;25;Hombre;vendedor
```

Registrar Clientes

Julieta Martinez

1751579654

julieta59@gmail.com

0995695895

De los Fresnos E6-49

Añadir

Registrar Clientes

Julieta Martinez

17515

juliet

0995

De los Fresnos E6-49

Añadir

Éxito  
i Cliente registrado correctamente  
OK

```
edores empleados.t client x + - □ ×
File Edit View
Ana Maria;1751578456;ana21@gmail.com;0995695895;la carolina
Julieta Martinez;1751579654;julieta59@gmail.com;0995695895;De los
Fresnos E6-49
```

registroVentas

REGISTRO DE VENTAS

Cedula del cliente: 1751579655

Nombre:

Dirección:

Teléfono:

Correo:

Empleado:

Producto:

Cantidad:

Advertencia

! Cliente no encontrado.

OK

Producto	Cantidad	Precio Unitario	Total
----------	----------	-----------------	-------

Subtotal:

IVA:

Total:

registroVentas

REGISTRO DE VENTAS

Cedula del cliente: 1751579654

Nombre:

Dirección:

Teléfono:

Correo:

Empleado:

Producto:

Cantidad:

Producto	Cantidad	Precio Unitario	Total
----------	----------	-----------------	-------

Subtotal:

IVA:

Total:

registroVentas

REGISTRO DE VENTAS


Cedula del cliente: 1751579654


Nombre: Julieta Martinez

Dirección: De los Fresnos E6-49

Teléfono: 0995695895

Correo: julieta59@gmail.com

Empleado:  Advertencia

Producto:  Stock insuficiente para este producto.

Cantidad:

	Producto	Cantidad	Precio Unitario	Total
1	monitor	5	120	600

Subtotal: 600

IVA: 90

Total: 690

registroVentas

REGISTRO DE VENTAS

Cedula del cliente: 1751579654

Nombre: Julieta Martinez

Dirección: De los Fresnos E6-49

Teléfono: 0995695895

Correo: julieta59@gmail.com

Empleado: Julian Torres

Producto: Parlantes

Cantidad: 4

	Producto	Cantidad	Precio Unitario	Total
1	monitor	5	120	600
2	Parlantes	4	58	232

Subtotal: 832

IVA: 124.8

Total: 956.8

registroVentas

### REGISTRO DE VENTAS

Cedula del cliente: 1751579654 Cargar

Nombre: Julieta Martinez

Dirección: De los Fresnos E6-49

Teléfono: 0995695895

Correo: julieta59@gmail.com

Empleado:

Producto:

Cantidad:

Éxito

Venta registrada exitosamente.

OK

	Producto	Cantidad	Precio Unitario	Total
1	monitor	5	120	600
2	Parlantes	4	58	232

Subtotal: 832

IVA: 124.8

Total: 956.8

PAGAR

```

users.txt | proveedores | empleados.t | clientes.txt | productos.tx | venta
File Edit View
1751579654,Julieta Martinez,De los Fresnos E6-49,0995695895,julieta59@gmail.com,Julian
Torres
monitor,5,120,600
Parlantes,4,58,232

```

Problemas detectados durante el desarrollo de la práctica/deber