Índice

Introducción

El proyecto **ESIZON** consiste en un una versión simplificada de una aplicación de envíos de compra/venta productos que consta de unos ficheros donde se guarda la información, relacionada tanto con los pedidos y productos como con los datos del usuario. En la aplicación se podrán realizar adquisiciones de pedidos como clientes, o ventas como proovedor, ya que la aplicación permite la gestión de ventas de productos a empresas externas. Además contará, con descuentos para pedidos, o una red de transportistas, e incluso ESIlockers para entregas de los productos solicitados. Todo lo registrado en la app, contiene un ID, que permite reconocer todo más fácilmente.

Documentación de usuario

En cuanto a las opciones de usuario, los clientes tendrán desde la opción de inicio de sesión, usual en cualquier aplicación. Hasta la posibilidad de ordenar productos, buscándolos por su categoría, nombre o incluso descripción. Además tendra una cartera virtual, y además podrá solicitar que las entregas de sus pedidos sean a su propio domicilio o a un ESILockers.

Descripción funcional

- Debe describir de forma simple el propósito del sistema: El propósito principal del sistema es facilitar el proceso de compra y venta de productos, vía online, tanto a clientes como a administradores. Permitiendo explorar una variedad de categorías de productos, y realizar pedidos de estos. Además se permitirá a cada tipo de usuario gestionar sus cuentas. Por otra parte, se otorgan las herramientas necesarias a los administradores, pudiendo gestionar productos, pedidos, o acceder a la información del resto de usuarios, también podrán proporcionar descuentos a los diferentes productos.

- En esta sección se debe incluir una breve descripción funcional sobre lo que hace y no hace el sistema: El sistema abarca con la gran mayoría de operaciones esenciales para facilitar adquisiciones de productos en línea. El sistema incluye desde funciones como:

    - Gestión de Clientes: Registros, inicio de sesion, y gestión de cuentas.
    - Gestión de Productos: Desde el rol de cliente se podrá seleccionar los productos que se quiere comprar. En cambio los administradores y proveedores externos, podrán añadir, eliminar productos del catálogo, y poder realizar un seguimiento del inventario.
    - Gestión de Pedidos: Creación, modificación y cancelación de pedidos, además de un seguimiento del estado de estos.
    - Gestión de Descuentos: Aplicación de descuentos a los productos individuales o a pedidos completos.

Hay que mencionar también aquello que no incluye el sistema,

- Hay que tener en cuenta, que aunque el sistema simula una aplicación de comercio en línea, no incluye un sistema de pago oficial, sino que consta de una simulación de economía.
- No contiene capacidades avanzadas de análisis predictivo.
- No posee acceso a sistemas externos.

Tecnología

El sistema esta escrito en lenguaje C, se han utilizado ficheros tipo .txt, para almanecer los distintos tipos de datos. A la hora de programar, se han usado programas como: **Visual Studio Code** y **Codeblocks**. Además como programa vínculo de enlace entre los distintos códigos de cada programador, se ha usado **Github**.

Dentro del propio sistema, para el funcionamiento de distintas funciones se han usado librerias como: *time.h*, *windows.h* o *string.h*, además que para cada módulo se ha creado su respectiva librería, con los registros y funciones correspondientes.

```
void fecha_entrega(char *fecha, int dia_entrega)
{

    time_t rawtime; // Defino la struct de tiempo.

    // Convertir la fecha actual a time_t
    time(&rawtime);

    // Sumar x días a la fecha actual.
    rawtime += dia_entrega * 24 * 60 * 60; // X días en segundos
```

```
    // Actualizar la estructura tm
    struct tm *nueva_fecha = localtime(&rawtime);

    strftime(fecha, 11, "%d/%m/%Y", nueva_fecha); // Convierte la struct tm en un
formato legible tal que dia/mes/año
}
```

- Code::Blocks: Version 20.3
- Visual Code Blocks: Version 1.88.1
- Github

Acceso al sistema

Al iniciar el programa, se ejecutará la función ***accederPrograma()***, que se encarga del inicio al sistema.Una vez se ha accedido, se carga la función ***iniciarSesison()***. Tras el inicio de sesión, se comprobará que el usuario identificado existe, en tal caso, se cargarán funciones que muestrarán la distinta información con respecto al rol del usuario. Funciones como ***menuadmin()***, ***menutransportista()***, ***menuusuario()*** y ***menutransportista()***. Para salir del sistema, simplemente el usuario seleccionará la opción *Salir del Sistema*.

> Ejemplo de menú de inicio, para Rol de Transportista

```
// Rol de transportista
// Solo puede acceder: Transportista
void menutransportista(int rol, int posVectorClienteActual, Locker **lockers,int
*numLockers ,Cliente **clientes, int *numClientes, AdminProv **adminProvs, int
*numAdminProvs,
                       Transportista **transportistas, int *numTransportistas,
Producto **productos, int *numProductos, Categoria **categorias, int
*numCategorias, int *numProductoPedidos, ProductoPedido **productoPedidos,
CompartimentoLocker **compartimentoLockers, int *numCompartimentoLockers)
{
    int opcion;
    char id_t[7]; // Incrementamos el tamaño para incluir el carácter nulo

    do
    {
        printf("------------------------------------------------------\n");
        printf("             NOMBRE: %s                                \n",
(*transportistas)[posVectorClienteActual].nombre);
        printf("------------------------------------------------------\n");
        printf("|1-Perfil                                             |\n");
        printf("|2-Repartos                                           |\n");
        printf("|3-Retornos                                           |\n");
        printf("|4-Salir del sistema                                  |\n");
        printf("------------------------------------------------------\n");
        scanf("%d", &opcion);
        getchar(); // Consumir el carácter de nueva línea residual del búfer de
```

```
entrada

        switch (opcion)
        {
        case 1:
            perfil_t(&(*transportistas)[posVectorClienteActual]);
            break;
        case 2:
            reparto(productoPedidos, numProductoPedidos, (*transportistas)
[posVectorClienteActual].id_transp);
            enReparto(productoPedidos, numProductoPedidos, compartimentoLockers,
numCompartimentoLockers, id_t);
            break;
        case 3:
            retornoProducto(lockers,productos,numProductos, numLockers, (*lockers)
[posVectorClienteActual].localidad);
            break;
        case 4:
            printf("Hasta pronto, ESIZON\n");
            salirprograma();
            break;
        default:
            printf("Opción no válida\n");
            break;
        }
    } while (opcion != 4);
}
```

Los problemas a mencionar, pueden ser, que el usuario no escoja entre las opciones indicadas, para que cumpla dentro de lo previsto el sistema se encargará de que el usuario introduzca entre lo correspondiente. Asimismo, en caso de que el usuario sea de un rol menor, como cliente, y trate de realizar una función que sea de otro rol, el sistema actuará como en el anterior caso, e indicará al usuario que lleve a cabo una acción dentro de lo permitido.

Manual de referencia

La aplicación ESIZON, ofrece múltiples ventajas tanto como para sus clientes como para los administradores. Entre otras destacan:

- El acceso las 24 horas, al ser una aplicación web de compra ventas de artículos, se podrán realizar adquisicion y ventas en cualquier momento del día.
- La aplicación ofrece un amplio catálogo de productos, y por lo tanto, de categorías. Amoldándose a las diferentes demandas que puedan tener los clientes que ingresen en la aplicación.
- Una gestión propia de envíos, la aplicación consta de su propio sistema de transportista que llevan los pedidos de la manera más eficiente posible. Además la aplicación tiene su propia red de **lockers**, que permite una entrega más segura de los pedidos, sino se desea una entrega en el domicilio.

El uso del sistema, dependerá del usuario:

- Los clientes podrán escoger entre si desea modificar datos de su perfil, adquerir productos, gestionar los descuentos que posee, hacer y ver el estado de sus pedidos y/o devoluciones.

- Los administradores, podrán dar de alta productos, pedidos, descuentos, categorias, etc... . Además podrán gestionar también los productos, pedidos, clientes, proveedores, descuentos, devoluciones.
- Los proveedores, tendrán una función parecida a los administradores, pero estos únicamente tendrán permisos en los productos que le pertenezcan.
- Los transportistas, usarán la app para actualizar el estado de los pedidos, y observar los sitios de entrega (Lockers).

Como cada uno de estos tienen distintos permisos, al iniciar sesión deben ser identificados. Y contarán con diferentes menús.

Documentación del sistema

El proyecto **Esizon** está organizado en múltiples archivos de tipo "*.c*" y archivos de encabezado "*.h*", cada uno de ellos contiene la implementación de los distintos módulos del sistema. También, hay que mencionar el archivo ***Utilidades.c*** que consta de distintas funciones que se han repetido durante el desarrollo del sistema. Como la función de depuración del buffer, para así evitar saltos de línea no deseados o la función de obtener la fecha actual.

```c
#include <stdio.h>
#include <time.h>


void flushInputBuffer(){
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

void obtener_fecha_actual(char *fecha_actual)
{
    time_t rawtime;
    struct tm *info;
    time(&rawtime);
    info = localtime(&rawtime);
    strftime(fecha_actual, 11, "%d/%m/%Y", info);
}
```

Se han utilizado ficheros, para almacenar la información relacionada con cada dato del sistema introducido. La división para los distintos ficheros es la misma que se ha realizado para los módulos:

- Módulo de Productos
- Módulo de Pedidos
- Módulo de Clientes
- Módulo de Proveedores
- Módulo de Descuentos
- Módulo de Transportistas
- Módulo de Devoluciones
- Módulo de Lockers

- Módulo de CompartimentosLockers
- Módulo de Categorías
- Módulo de Administración/Proveedores

Se han definido estructuras de datos para cada componente del sistema, que representan la información almacenada en los ficheros. Estas estructuras incluyen miembros que representan los atributos relevantes de cada entidad, como nombre, descripción, precio, etc... .

El código tiene una parte externa, que es el archivo llamado ***"ESIZON.c"*** donde se lleva a cabo el inicio de sesión y se muestra los distintos menús, dentro de este se hará llamadas a gran parte de funciones de los distintos módulos. De este modo habrá módulos más dependientes de otro, como es el caso de ProductosPedidos, ya que en este se necesitará saber sobre productos o pedidos entre otros. Y el caso contrario de categoría que prácticamente es un módulo que no depende de otros. En los siguientes fragmentos podremos observar la distinta dependencia de dos módulos trabajados.

> Archivo cabecera de Productos_pedidos.h

```
#ifndef Producto_Pedido_H
#define Producto_Pedido_H

#include "Cliente.h"
#include "Utilidades.h"

#include "Productos.h"
#include "Pedidos.h"
#include "Lockers.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    char id_pedido[8];
    char id_prod[8];
    int num_unid;
    char fecha_entrega_prevista[11];
    float importe;
    char estado_pedido[15]; // "enPreparacion", "enviado", "enReparto",
"enLocker", "entregado", "devuelto", "transportista"
    char id_transp[5];
    char id_locker[11];
    char cod_locker[11];
    char fecha_entrega_devolucion_transp[11];
}ProductoPedido;

void baja_prodPed(ProductoPedido **pedidos , int* tamanioProP, char *id_baja);
void listado_pedido(ProductoPedido **pedidos, int* tamanio);
void buscarPedido (ProductoPedido **pedidos, int** tamanio , char* id);
void consultar_estado(ProductoPedido *pedidos, int* tamanio, char
*id_pedido_buscado);
char *seleccionar_producto(Producto *productos, int* tamanio, char
```

```
*productos_select);
int reducirStock(Producto *productos, int tamProd,char*idProd, int ctdadReducir);
void guardarProductoPedidoEnArchivo(ProductoPedido *, int );
ProductoPedido *iniciarProductoPedidosDeArchivo(int *);
void hacerPedido(Producto **listaCompra, int *tamLista,Producto **productos, int
*numnProductos,
                 Pedido **pedidos, int *numPedidos, ProductoPedido **prodPeds,int
*numProdPe, char *idCliente,
                  char *tipoEntrega, char *idLocker);
void listado_pedido_pendiente(ProductoPedido **pedidos, int* tamanio, char
*idProd,Producto **productos, int * nProductos);


#endif
#endif
```

Archivo cabecera de categorías.h

```
#ifndef CATEGORIA_H
#define CATEGORIA_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Utilidades.h"


typedef struct {
    char id_categ[5];
    char descrip[51];
}Categoria;


//En las altas y bajas habra que añadir id cliente dependiendo sea proveedor o
admin o cliente.
char* id_generator_categ(Categoria *, int );
void alta_categoria(Categoria **categoria, int* tamanio_vector);
void baja_categoria(Categoria **categorias, int *numCategorias);
int check_categ(Categoria **, int *,char *);
char *indicar_categ(Categoria **, int* , char *);
void modificarCategoria(Categoria **categorias, int *numCategorias);
void guardarCategoriasEnArchivo(Categoria *categorias, int numCategorias);
Categoria *iniciarCategoriasDeArchivo(int *numCat);
void listarCategorias(Categoria **categorias, int *numCategorias);
void buscarCategoria(Categoria **categorias, int *numCategorias);

#endif
```

Especificación del sistema

1. Descomposición del Problema:

Durante la fase de diseño del programa, se dividió el proyecto en distintos subproblemas que formaban parte del sistema global. Estos subproblemas se desglosaron en módulos independientes, cada uno de los cuales se encargaría de abordar una parte específica del problema general. Los subproblemas identificados incluyeron la gestión de productos, pedidos, clientes, proveedores, descuentos, transportistas, devoluciones, lockers, compartimentos de lockers y categorías.

2. Módulos Asociados:

Cada subproblema se asignó a un módulo específico del sistema, que sería responsable de su implementación y funcionamiento. Los módulos asociados incluyeron:

- Módulo de Productos
- Módulo de Pedidos
- Módulo de Clientes
- Módulo de Proveedores
- Módulo de Descuentos
- Módulo de Transportistas
- Módulo de Devoluciones
- Módulo de Lockers
- Módulo de CompartimentosLockers
- Módulo de Categorías
- Módulo de Administración/Proveedores

3. Especificación de Módulos:

Cada módulo se especificó detalladamente, describiendo su funcionalidad, los datos que manejaría, las operaciones que realizaría y las interfaces de usuario que proporcionaría. Se establecieron interfaces claras entre los diferentes módulos para facilitar la comunicación y la integración del sistema.

4. Plan de Desarrollo del Software:

El desarrollo del software se llevó a cabo de manera iterativa y colaborativa, con reuniones regulares del equipo para discutir avances, problemas y mejoras. Se estableció un plan de desarrollo que incluyó la asignación de tareas, los plazos de entrega, las pruebas a realizar y los hitos importantes del proyecto. Se utilizó un enfoque ágil de desarrollo de software, lo que permitió adaptar el proceso según las necesidades y los cambios en los requisitos del proyecto.

5. Revisiones y Mejoras:

Se realizaron revisiones periódicas de cada módulo del sistema para identificar fallos y áreas de mejora. Se fomentó la retroalimentación entre los miembros del equipo para compartir conocimientos, resolver problemas y optimizar el código y la funcionalidad del sistema.

Módulos

Cada módulo, se encarga de ir cargando los datos que les corresponde en los ficheros convenientes para el funcionamiento del programa. Además cada módulo posee una función que se encarga de generar un ID

único, para cada tipo de dato que corresponde al módulo en el que se encuentre.

- **Módulo de Productos:**

Este módulo se encarga de gestionar toda la información relacionada con los productos disponibles en la plataforma de comercio en línea. Incluye la altas, modificación y bajas de productos, así como la búsqueda y visualización de información detallada sobre cada uno de ellos, como nombre, descripción, precio, categoría, etc.

- **Módulo de Pedidos:**

El módulo de pedidos se encarga de administrar los pedidos realizados por los clientes. Permite la creación y gestión de pedidos, la asociación con clientes y productos, el seguimiento del estado del pedido y la generación de informes relacionados con las ventas.

- **Módulo de Clientes:**

Este módulo gestiona la información de los clientes que utilizan la plataforma. Permite la creación y actualización de perfiles de clientes, la gestión de direcciones de envío y facturación, así como la visualización del historial de pedidos y la aplicación de descuentos personalizados.

- **Módulo de Proveedores:**

El módulo de proveedores administra la información de los proveedores de productos. Permite la creación y gestión de perfiles de proveedores, la asociación con productos suministrados, la actualización de datos de contacto y la gestión de relaciones comerciales.

- **Módulo de Descuentos:**

Este módulo se encarga de gestionar los descuentos disponibles en la plataforma. Permite la creación y aplicación de descuentos a productos específicos o a pedidos completos.

- **Módulo de Transportistas:**

El módulo de transportistas administra la información de las empresas de transporte asociadas al sistema. Permite la gestión de perfiles de transportistas, la asignación de envíos a transportistas específicos y el seguimiento del estado de los envíos.

- **Módulo de Devoluciones:**

Este módulo gestiona el proceso de devoluciones de productos por parte de los clientes. Permite la solicitud y aprobación de devoluciones, la gestión de reembolsos o cambios de productos, y el seguimiento del estado de las devoluciones.

- **Módulo de Lockers:**

El módulo de lockers administra la información de los casilleros de almacenamiento utilizados para las entregas de pedidos. Permite la gestión de la disponibilidad de lockers, la asignación de pedidos a casilleros específicos y la gestión de códigos de acceso.

- **Módulo de CompartimentosLockers:**

Este módulo gestiona la información detallada de los compartimentos individuales dentro de cada locker. Permite la asignación de productos a compartimentos específicos, el seguimiento del estado de los productos almacenados y la gestión de notificaciones de entrega.

- **Módulo de Categorías:**

El módulo de categorías administra la clasificación de productos en diferentes categorías o etiquetas. Permite la creación y gestión de categorías, la asociación de productos a categorías específicas y la navegación por categorías en la plataforma de venta.

- **Módulo de Administración/Proveedores:**

Este módulo proporciona herramientas de administración para gestionar el sistema en su conjunto y las relaciones con los proveedores. Incluye funciones de gestión de usuarios, configuración del sistema, control de acceso y análisis de datos. Además dentro de él se diferencia entre los administradores que tienen amplitud de permisos y los proveedores, que solo actuan sobre sus propios productos.

- **Módulo de ProductosPedidos:**

El módulo ProductosPedidos gestiona la relación entre productos y pedidos. Es responsable de asociar los productos incluidos en cada pedido, así como de mantener actualizada la información sobre la cantidad de productos solicitados en cada orden. Permite la actualización de inventario en función de los productos incluidos en los pedidos y facilita la generación de informes sobre la demanda de productos y la gestión de stocks.

Plan de prueba

Se han llevado a cabo pruebas exhaustivas del sistema para verificar su funcionamiento correcto y garantizar la integridad de los datos. Por otro lado, se han elaborado un número de pruebas en donde se incluyen pruebas de unidad, pruebas de integración y pruebas de aceptación para validar todas las funcionalidades del sistema.

Prueba de los módulos

Se han realizado pruebas en los distintos módulos para comprobar el funcionamiento adecuado de las funciones. En transportistas, se verifica que tanto las funciones altas y bajas, añadan datos y eliminen estos cuando corresponda. Para ello se realizaron pruebas como:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Transportista.h" // Suponiendo que este es el archivo que contiene las
definiciones de las funciones

int main() {
    Transportista *transportistas = NULL;
    int num_transportistas = 0;

    // Prueba de alta_transportista
```

```
    printf("=== Probando alta_transportista ===\n");
    printf("Ingrese datos del nuevo transportista:\n");
    alta_transportista(&transportistas, &num_transportistas);
    printf("Nuevo transportista agregado:\n");
    listado_transportista(transportistas, num_transportistas);

    // Prueba de baja_transportista
    printf("\n=== Probando baja_transportista ===\n");
    printf("Ingrese el ID del transportista a dar de baja: ");
    char id_baja[5];
    scanf("%4s", id_baja);
    flushInputBuffer(); // Suponiendo que tienes una función flushInputBuffer para
limpiar el buffer del teclado
    baja_transportista(transportistas, &num_transportistas, id_baja);
    printf("Transportista dado de baja:\n");
    listado_transportista(transportistas, num_transportistas);

    // Liberar memoria
    free(transportistas);

    return 0;
}
```

Por otro lado en el mismo módulo Transportistas se comprueba el correcto funcionamiento de la función **reparto**, con el objetivo de probar esta con diferentes números de pedidos para asegurarse de que todos los pedidos asignados al transportista se procesen correctamente.

```
#include <stdio.h>
#include <stdlib.h>
#include "Transportistas.c"

// Función para realizar pruebas de caja blanca
void test_reparto() {
    printf("Iniciando pruebas de reparto...\n");

    // Crear algunos pedidos de muestra
    ProductoPedido pedidos[] = {
        {"001", "prod001", "trans001", "enReparto", "10/05/2024", 10.5,
"locker001", ""},
        {"002", "prod002", "trans001", "enReparto", "12/05/2024", 15.25,
"locker002", ""},
        {"003", "prod003", "trans001", "enReparto", "14/05/2024", 20.0,
"locker003", ""}
    };

    int num_pedidos = sizeof(pedidos) / sizeof(pedidos[0]);

    // Simular reparto para cada pedido
    for (int i = 0; i < num_pedidos; i++) {
        printf("\nSimulando reparto para el pedido %d...\n", i + 1);
```

```
        reparto(pedidos, num_pedidos, "trans001");
    }

    printf("\nPruebas de reparto completadas.\n");
}

int main() {
    test_reparto();
    return 0;
}
```

También se han generado funciones que se encargan de comprobar los casos extremos y el que se cumpla el funcionamiento adecuado, de las funciones generadoras de IDs.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Transportistas.c"

// Función para realizar pruebas de caja blanca
void test_id_generator_trans() {
    printf("Iniciando pruebas de id_generator_trans...\n");

    // Prueba con un vector de transportistas vacío
    printf("\nPrueba con un vector de transportistas vacío:\n");
    Transportista *transportistas_vacio = NULL;
    char *id_vacio = id_generator_trans(transportistas_vacio, 0);
    printf("ID generada para vector vacío: %s\n", id_vacio);
    free(id_vacio);

    // Prueba con un vector de transportistas lleno (máximo número de elementos)
    printf("\nPrueba con un vector de transportistas lleno:\n");
    int max_transportistas = 1000;
    Transportista *transportistas_lleno = (Transportista
*)malloc(max_transportistas * sizeof(Transportista));
    if (transportistas_lleno == NULL) {
        printf("Error: No se pudo asignar memoria para el vector de transportistas
lleno.\n");
        return;
    }

    // Llenar el vector de transportistas con datos ficticios
    for (int i = 0; i < max_transportistas; i++) {
        snprintf(transportistas_lleno[i].id_transp,
sizeof(transportistas_lleno[i].id_transp), "%03d", i + 1);
    }

    // Generar ID para el vector lleno
    char *id_lleno = id_generator_trans(transportistas_lleno, max_transportistas);
    printf("ID generada para vector lleno: %s\n", id_lleno);
    free(id_lleno);
```

```
    free(transportistas_lleno);

    printf("\nPruebas de id_generator_trans completadas.\n");
}

int main() {
    test_id_generator_trans();
    return 0;
}
```

El siguiente código se encarga de hacer pruebas en cuanto al módulo de Productos.c, comprobando el funcionamiento en distintos casos extremos.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Productos.h"

void test_limites_productos() {
    // Crear un vector de productos con un tamaño fijo para simular un límite.
    int tamano_maximo = 5;
    Producto *productos = malloc(tamano_maximo * sizeof(Producto));
    if (productos == NULL) {
        printf("Error: No se pudo asignar memoria para el vector de
productos.\n");
        return;
    }

    // Llenar el vector con productos para alcanzar el límite.
    for (int i = 0; i < tamano_maximo; i++) {
        char id[8];
        sprintf(id, "%07d", i + 1); // Generar un ID único para cada producto
        strcpy(productos[i].id_prod, id);
    }

    // Intentar agregar un nuevo producto cuando el vector está lleno.
    printf("Intentando agregar un nuevo producto cuando el vector está lleno:\n");
    alta_producto(&productos, &tamano_maximo, "0000000", NULL, NULL); // Se pasa
un ID falso, ya que no se proporciona uno real en este ejemplo.
    printf("\n");

    // Eliminar todos los productos del vector.
    printf("Eliminando todos los productos del vector:\n");
    while (tamano_maximo > 0) {
        baja_producto(&productos, &tamano_maximo, productos[0].id_prod); // Se
elimina el primer producto en cada iteración.
    }
    printf("\n");

    // Intentar eliminar un producto cuando el vector está vacío.
```

```
    printf("Intentando eliminar un producto cuando el vector está vacío:\n");
    baja_producto(&productos, &tamano_maximo, "0000000"); // Se pasa un ID falso,
ya que no hay productos en el vector.
    printf("\n");

    // Crear un nuevo producto con el máximo número de caracteres en la
descripción.
    printf("Agregando un producto con la descripción más larga permitida:\n");
    Producto nuevo_producto;
    strcpy(nuevo_producto.id_prod, "0000001");
    strcpy(nuevo_producto.nombre, "Producto 1");
    strcpy(nuevo_producto.descrip, "Los productos con características físicas
únicas");
    strcpy(nuevo_producto.id_categ, "0001");
    strcpy(nuevo_producto.id_gestor, "0001");
    nuevo_producto.stock = 10;
    nuevo_producto.entrega = 5;
    nuevo_producto.importe = 19.99;
    printf("Producto agregado:\n");
    printf("ID: %s\nNombre: %s\nDescripción: %s\nCategoría: %s\nGestor: %s\nStock:
%d\nEntrega: %d\nImporte: %.2f\n",
           nuevo_producto.id_prod, nuevo_producto.nombre, nuevo_producto.descrip,
nuevo_producto.id_categ,
           nuevo_producto.id_gestor, nuevo_producto.stock, nuevo_producto.entrega,
nuevo_producto.importe);

    // Liberar la memoria utilizada por el vector de productos.
    free(productos);
}
```

Por otro lado, en el módulo clientes se realizó un main como sistema de gestión de perfiles de clientes. En él se comprueba el menú de usuario, ya una vez iniciado sesión. Además, el código también proporciona un bucle en el main que permite al usuario realizar múltiples acciones consecutivas.

```
int main()
{
    Cliente cli;
    int c,b,a = 0;

    printf("Nombre: ");
    flushInputBuffer();
    fgets(cli.nomb_cliente,21,stdin);

    printf("Provincia: ");
    fflush(stdin);
    fgets(cli.provincia,21,stdin);

    printf("Localidad: ");
    flushInputBuffer();
    fgets(cli.localidad,21,stdin);
    cli.localidad[strcspn(cli.localidad,"\n")] = '\0';
```

```
    printf("Direccion: ");
    flushInputBuffer();
    fgets(cli.dir_cliente,51,stdin);
    cli.dir_cliente[strcspn(cli.dir_cliente,"\n")] = '\0';

    printf("Email: ");
    flushInputBuffer();
    fgets(cli.email,31,stdin);

    printf("Contrasenia: ");
    flushInputBuffer();
    fgets(cli.contrasenia,16,stdin);

    printf("Saldo: ");
    flushInputBuffer();
    scanf("%f",&cli.cartera);
    printf("----------\n\n");

    while(a == 0)
    {
        printf("(1) Mostrar perfil\n");
        printf("(2) Modificar perfil\n");

        flushInputBuffer();
        scanf("%d",&b);

        switch(b)
        {
            case(1):
                mostrar_cliente(cli);
            break;

            case(2):
                c = corroborar_contrasenia(cli.contrasenia);

                if(c == 0)
                {
                    cambiar_perfil_cliente(&cli);
                }
                else
                {
                    printf("Contrasenia incorrecta\n");
                }
        }

        printf("Presione 0 para continuar pruebas:");
        flushInputBuffer();
        scanf("%d",&a);

        printf("----------\n\n");
    }
```

```
    return 0;
}
```

Prueba de integración

La función ***verificar_contrasenia***, dentro de proveedor, se encarga de comparar entre módulos, de este modo,verifica si la contraseña proporcionada por el usuario coincide con la contraseña almacenada en la estructura **AdminProv**. Utiliza *strcmp* para comparar las contraseñas y devuelve un valor entero que indica si las contraseñas coinciden.

```
int verificar_contrasenia(char contrasena[16])// para modificar los datos del
usuario se pedira la contrasenia
{
    char verificacion[16];
    int a;

    printf("Contrasena: ");
    flushInputBuffer();
    fgets(verificacion,16,stdin);// Se introduce la contrasenia

    a = strcmp(verificacion,contrasena); // Compara si la contrasenia introducida
y la del usuario son iguales

    return a; //si devuelve 0 el usuario ha introducido la contrasenia correcta
}
```

La función ***hacerPedido()*** interactúa con los módulos de pedidos **(ProductoPedido.h)** y productos **(Producto.h)** para realizar un nuevo pedido. Utiliza funciones como *reducirStock()* para actualizar el stock de productos después de realizar un pedido. También puede interactuar con otros módulos dependiendo de la implementación específica de las funciones llamadas dentro de *hacerPedido()*.

```
void hacerPedido(Producto *listaCompra, int tamLista,Producto *productos, int
numnProductos, Pedido **pedidos, int *numPedidos, ProductoPedido **prodPeds,int
*numProdPe, char *idCliente, char *tipoEntrega, char *idLocker, float *costeTotal)
{

    // CREO EL NUEVO PEDIDO
    Pedido nuevo;

    strcpy(nuevo.id_pedido, id_generator_pedido(*pedidos, *numPedidos));
    obtener_fecha_actual(&nuevo.fecha_pedido);
    strcpy(nuevo.lugar_entrega, tipoEntrega);
    strcpy(nuevo.id_locker, idLocker);
    strcpy(nuevo.id_cliente, idCliente);

    *pedidos = (Pedido *)realloc(*pedidos, (*numPedidos + 1) * sizeof(Pedido));
    (*pedidos)[*numPedidos] = nuevo;
    (*numPedidos)++;
```

```
    float coste = 0;
    // CREO LOS PRODUCTOSPEDIDOS
    for (int i = 0; i < *numProdPe; i++)
    {
        ProductoPedido prodPed;
        strcpy(prodPed.id_pedido, nuevo.id_pedido);
        strcpy(prodPed.id_prod, listaCompra[i].id_prod);

        do{
        printf("Cuantas unidades quieres de %s:", listaCompra[i].nombre);
        scanf("%d", &prodPed.num_unid);
        flushInputBuffer();
        }while(prodPed.num_unid <= 0 ||
reducirStock(productos,&numPedidos,prodPed.id_prod,prodPed.num_unid)!=0); //

        prodPed.importe = prodPed.num_unid * listaCompra[i].importe;
        coste += prodPed.importe;
        strcpy(prodPed.estado_pedido, "enPreparacion");
        strcpy(prodPed.id_locker, idLocker);

        *prodPeds = (ProductoPedido *)realloc(*prodPeds, (*numProdPe + 1) *
sizeof(ProductoPedido));
        (*prodPeds)[*numProdPe] = prodPed;
        (*numProdPe)++;
    }

    (*costeTotal) = coste;
}
```

Plan de pruebas de aceptación

Comprar un producto:

- **Descripción:** Esta prueba verifica que un usuario pueda realizar una compra exitosa de un producto desde la plataforma.
- Pasos a seguir:
  - Iniciar sesión en la plataforma como usuario registrado.
  - Buscar un producto deseado.
  - Agregar el producto al carrito de compras.
  - Proceder al pago y completar la transacción.
- **Resultados esperados:** El producto se agrega al carrito de compras y se reduce el stock disponible del producto. Se completa la transacción de compra correctamente.

Añadir un producto como proveedor:

- **Descripción:** Esta prueba verifica que un usuario con privilegios de proveedor pueda agregar un nuevo producto al sistema.
- Pasos a seguir:
  - Iniciar sesión en la plataforma como proveedor.

    - Acceder al panel de control de proveedores.
    - Agregar un nuevo producto proporcionando todos los detalles necesarios (nombre, descripción, precio, etc.).
- **Resultados esperados**: El nuevo producto se agrega correctamente al sistema y está disponible para su compra en la plataforma.

Reducir stock de los productos cuando son comprados:

- **Descripción:** Esta prueba verifica que el stock de un producto se reduzca automáticamente cuando se realiza una compra.
- Pasos a seguir:
    - Verificar el stock disponible de un producto antes de realizar una compra.
    - Comprar el producto deseado.
    - Verificar nuevamente el stock disponible del producto después de la compra.
- **Resultados esperados:** Después de la compra, el stock disponible del producto se reduce en la cantidad correspondiente al número de unidades compradas.

Actualizar la información del producto:

- **Descripción:** Esta prueba verifica que un usuario autorizado pueda actualizar la información de un producto existente.
- Pasos a seguir:
    - Iniciar sesión en la plataforma con privilegios de administrador o proveedor.
    - Buscar y seleccionar el producto que se desea actualizar.
    - Modificar los detalles del producto, como su nombre, descripción, precio, etc.
    - Guardar los cambios y verificar que se hayan aplicado correctamente.
- **Resultados esperados:** Los cambios realizados en la información del producto se guardan correctamente en el sistema y se reflejan en la plataforma para los usuarios.

Documentación del código fuente

La documentación interna ha sido generada con Doxygen, para acceder a ella pulse aquí.

Referencias

Se han tomado de referencia, información sacada de distintos foros, como stack_overflow o tutorialspoint.