**CENTRO UNIVERSITÁRIO CAMPOS DE ANDRADE**

**TECNOLOGIA EM CIÊNCIAS DA COMPUTAÇÃO**

**ESTOQUE DE UM SALÃO DE BELEZA**

**Versão 1.0**

**2023**

Douglas Nascimento

Pablo Celestino da Rocha Lobo

André Henrique Fiatkoski Lustosa

**CONTROLE DE ESTOQUE**

Este documento tem o objetivo de apresentar o Relatório da disciplina de Algoritmos e Linguagem Estruturada do Centro Universitário Campos de Andrade – Uniandrade.

Orientador: Profº. Me. Edgar Cabral

CURITIBA

2023

**Sumário**

[1. INTRODUÇÃO 4](#_Toc131613478)

[1.1. Tema 4](#_Toc131613479)

[1.2. Objetivos do projeto 4](#_Toc131613480)

[2. DESCRIÇÃO GERAL DO SISTEMA 5](#_Toc131613481)

[2.1. Diagnose 5](#_Toc131613482)

[2.2. Regras de Negócio 5](#_Toc131613483)

[3. REQUISITOS DO SISTEMA 6](#_Toc131613484)

[3.1. Ambiente de Desenvolvimento 6](#_Toc131613485)

3.1.1. DEV C++ 6

3.1.2. VSCodium 6

3.1.3. GIT 6

[3.2. Bibliotecas 6](#_Toc131613486)

[3.3. Funções 6](#_Toc131613487)

[4. CÓDIGO FONTE 7](#_Toc131613488)

[5. CONSIDERAÇÕES FINAIS 8](#_Toc131613489)

[BIBLIOGRAFIA 9](#_Toc131613490)

## INTRODUÇÃO

### **Tema**

O projeto tem o tema de criar um sistema de controle de estoque na linguagem C para um salão de beleza, que envolve cabeleireiros, manicures e entre outros profissionais

### **Objetivos do projeto**

Este projeto tem objetivo de controlar um estoque de um salão de beleza, podendo se adicionar, remover ou verificar os produtos de um banco de dados, para que se verifique caso tenha falta, evitar o vencimento e fazer a remoção de produtos vencidos que podem causar danos a saúde de seus clientes. No final, podendo prejudicar o salão judicialmente e financeiramente.

## DESCRIÇÃO GERAL DO SISTEMA

### **Diagnose**

O salão de beleza Marcelus é localizado no bairro Fazendinha (Curitiba), numa rua comercial com vários estabelecimentos. O salão é composto de 3 cabeleireiros(as) e 2 manicures. Todos os colaboradores gerenciam o estoque dos produtos.

Temos objetivo de fazer um sistema que faça uma verificação da quantidade e validade dos produtos, pois essa verificação é feita manualmente.

A falta de produtos prejudicará, o trabalho dos colaboradores por não ter matéria-prima e diminuição da venda de produtos home-care.

Também pois produtos vencidos poderão ter resultados indesejados ou reações químicas prejudiciais até mesmo na saúda das pessoas.

### **Regras de Negócio**

### Os produtos são comprados através de distribuídoras oficiais das marcas

### O salão utiliza a forma de controle de estoque, o PEPS e PVPS. O PEPS significa que, quando o produto entra no estoque, já é posto nas prateleiras na frente para ser vendido. O PVPS é quando o produto que está mais perto de vencer tem prioridade para ser utilizado ou vendido primeiro.

* Controlar a movimentação dos produtos para uso interno e linha home-care (revenda) é extremamente importante para o controle de estoque. com essas informações, você pode fazer o planejamento de compras mais eficientes. Durante a entrada dos produtos é necessária que seja feita uma conferência rigorosa das mercadorias. Fazendo isso, deve-se conferir as quantidades e as datas de validade.
* O local de armazenamento deve ser em um local arejado., fora do alcance da luz solar e em temperatura ambiente.
* Classificar o estoque é saber **quais produtos de maior uso interno e venda**. Isso é importante para que o salão saiba quais itens devem ser repostos com frequência, quais precisam serem melhor trabalhados ou no seu uso ou na sugestão de venda

## REQUISITOS DO SISTEMA

### **Ambiente de Desenvolvimento**

### **DEV C++** (Versão 5.11)

### É uma IDE focado para o desenvolvimento de software na linguage C/C++. (Disponível somente no Windows)

### **VSCodium** (Versão 1.73.0)

### Essa IDE é contruído na base do Visual Studio Code, tendo a diferença de ser totalmente Libre, ou seja, sem partes de códigos proprietários e telemetria da Microsoft. E se pode usar para desenvolver quaisquer linguagens, só precisa ter o interpretador e compilador instalado na máquina. (Disponível para Linux, Windows e Mac)

### **GIT** (Versão 2.34.0)

### É uma ferramenta de controle de versões, ou seja o código pode passar por diferentes modificações, ou pode-se reverter essas modificaçôes, sem modificar o código principal localizado em um repositório remoto (GitLab e GitHub por exemplo).

### **Bibliotecas**

* **stdio.h**

Estão localizadas as funções das operações nas quais os mecanismos operam em função da entrada e da saída, bem como em arquivos também.

* **stdlib.h**

Possuí as funções responsáveis pela manipulação da alocação de memória, e da desalocação também

* **string.h**

Essa bibliotera é utilizado para a manipulação de strings, que são vetores de chars. Essas funções são capazes de fazer a contagem, cópia e concatenação, comparação e diversas outras funções.

* **time.h**

Possuí funções, variáveis para manipulação de unidades de tempo.

* **ctype.h**

As funções desta biblioteca serve para classificar e manipular caracteres ASCII, ou seja, para dizer se um dado byte representa uma letra, ou um dígito, ou um branco etc. em código ASCII

### **Funções**

**// Função para vallidar as datas de vencimento do produto na hora de cadastrar**

int dateExp(struct tm dateE);

**// Substituto do fflush(stdin)**

void cleanStdin(void);

**// Função para ler o arquivo inventory.dat**

void readFile(Tproduct inventory[], int \*size);

**// Função para escrever os dados no arquivo inventory.dat**

void writeFile(Tproduct inventory[], int size);

**// Função da busca binária**

int binarySearch(Tproduct inventory[], int key, int size);

**// Função auxiliar que verfica se o vetor está vazio**

int null(int size);

**// Função para ordernar os produtos que estão dentro de um vetor**

void sortProduct(Tproduct inventory[], int size);

**// Função para adicionar um novo produto**

void addProdutct(Tproduct inventory[], int \*size);

**// Função para exibir um relatório geral dos produtos cadastrados**

void generalReport(Tproduct inventory[], int size);

**// Função para exibir um relatório dos preços dos produtos**

void reportPriceList(Tproduct inventory[], int size);

**// Função para alterar dados de um produto**

void changeProduct(Tproduct inventory[], int size);

**// Função para deletar dados de um produto**

void deleteProduct(Tproduct inventory[], int \*size);

**// Função para achar um produto através de seu código**

void consultProduct(Tproduct inventory[], int size);

**// Função para achar um produto através ddo seu nome**

int consultProductByName(Tproduct inventory[], int size);

**// Função para o menu de relatórios especias**

void espMenu(Tproduct inventory[], int size);

**// Função para mostrar produtos vencidos (Não consegui terminar)**

void reportExpiredProduct(Tproduct invetory[], int size);

**// Função para aumentar o preço de compra dos produtos de um fornecedor**

void increaseSupplierPrices(Tproduct inventory[], int size);

**// Função para mostrar um relatório de produtos abaixo da quantidade mínima do estoque**

void reportMinimalInventory(Tproduct inventory[], int size);

**// Função para alterar as datas de vencimento do produto (outro que não deu certo)**

// void changeExpiredMenu(Tproduct inventory[], int size);

**// Função para alterar TODAS as informações de um produto (Não conseguimos esse também)**

// void changeFunction(Tproduct inventory[], int size);

## CÓDIGO FONTE

**#**include <stdio.h>

**#**include <stdlib.h>

**#**include <string.h>

**#**include <time.h>

**#**include <ctype.h>

const int MAX = 50; // Limiting the array size

int sortData = 0; // Variable to sort the data

// Type of a special data (Resgistry)

typedef struct Tproduct{

long int productCode;

char name[41];

int productQuantity;

float productBuyPrice;

int productGroup;

char productDescription[41];

char productUnit[3];

char productSupplier[41];

float productSellPrice;

float productMinimalProfit ;

int inventoryMin;

struct tm expiredDate;

} Tproduct;

// Function for validating the expiring date

int dateExp(struct tm dateE);

// Substitute for cleanStdin()

void cleanStdin(void);

// Struct for adding a expiration date for a product

struct tm getExpiredDate(){

struct tm dateE;

do {

printf("\nExpiring date of a product.\n");

printf("Day: ");

scanf("%d", &dateE.tm\_mday);

printf("\nMonth: ");

scanf("%d", &dateE.tm\_mon);

printf("\nYear: ");

scanf("%d", &dateE.tm\_year);

cleanStdin();

} while (!dateExp(dateE));

return dateE;

}

// Function for the reading of the inventory.dat file

void readFile(Tproduct inventory[], int \*size);

// Function for writing to the inventory.dat file

void writeFile(Tproduct inventory[], int size);

// Function for binary search

int binarySearch(Tproduct inventory[], int key, int size);

// Auxiliary function to verifie if the array is empty

int null(int size);

// Function to sort the products array per product code

void sortProduct(Tproduct inventory[], int size);

// Function to add a new product

void addProdutct(Tproduct inventory[], int \*size);

// Function to exhibit the general report of registered products

void generalReport(Tproduct inventory[], int size);

// Function to exhibit a report of the prices of registered products

void reportPriceList(Tproduct inventory[], int size);

// Function to make a change in a alredy existing product

void changeProduct(Tproduct inventory[], int size);

// Function to delete a product from the inventary

void deleteProduct(Tproduct inventory[], int \*size);

// Function to search for a specific product

void consultProduct(Tproduct inventory[], int size);

// Function to search for a specific product

int consultProductByName(Tproduct inventory[], int size);

// Function for menu of special reports

void espMenu(Tproduct inventory[], int size);

// Funtion to show a menu of reports for expired products

void reportExpiredProduct(Tproduct invetory[], int size);

// Funtion to increase the buy price of products from a specific supplier

void increaseSupplierPrices(Tproduct inventory[], int size);

// Function to show a report of products that are bellow the minimal\

quantity in the inventory

void reportMinimalInventory(Tproduct inventory[], int size);

void listReport(Tproduct invetory[], int index);

// Function to change the expiration dates of a specific product

// void changeExpiredMenu(Tproduct inventory[], int size);

// Function to change all values of a specific product

// void changeFunction(Tproduct inventory[], int size);

// Main function

int main(){

Tproduct inventory[MAX];

struct tm \*date;

time\_t dayToday;

time(&dayToday);

date = localtime(&dayToday);

int size=0, opt;

// Open the file of the data base

readFile(inventory, &size);

do{

printf("Today is: %d/%d/%d\n", date->tm\_mday, date->tm\_mon + 1, \

date->tm\_year + 1900);

printf("================================\n");

printf(" 1 - REGISTER PRODUCT\n");

printf(" 2 - CHANGE PRODUCT DETAILS\n");

printf(" 3 - DELETE PRODUCT\n");

printf(" 4 - SEARCH PRODUCT\n");

printf(" 5 - SEARCH PRODUCT BY NAME\n");

printf(" 6 - PRICE LIST REPORT\n");

printf(" 7 - GENERAL REPORT\n");

printf(" 8 - SPECIAL REPORTS\n");

printf(" 0 - EXIT\n");

printf("\nCHOOSE THE DESIRED OPTION:\n");

scanf("%d",&opt);

system("clear");

switch(opt){

case 1:

addProdutct(inventory, &size);

writeFile(inventory, size);

break;

case 2:

changeProduct(inventory, size);

writeFile(inventory, size);

break;

case 3:

deleteProduct(inventory, &size);

writeFile(inventory, size);

break;

case 4:

consultProduct(inventory, size);

break;

case 5:

consultProductByName(inventory, size);

break;

case 6:

reportPriceList(inventory, size);

break;

case 7:

generalReport(inventory, size);

break;

case 8:

espMenu(inventory, size);

break;

case 0:

printf("Thanks for using our software!!\n");

exit(0);

default:

printf("INVALID OPTION!!\n\n");

system("clear");

}

}while(opt!=0);

// Writing to the file after finishing the opperation

writeFile(inventory, size);

return 0;

}

void readFile(Tproduct inventory[], int \*size){

FILE \*file;

file = fopen("inventory.dat", "a+b");

// Open to attach to the file

if (!file){

printf("Error to open file!");

return;

}

// Read the data of the file

while(!feof(file)){

fread (&inventory[\*size], sizeof (Tproduct) , 1, file);

(\*size)++;

}

(\*size)--;

// Close the file

fclose(file);

return;

}

void writeFile(Tproduct inventory[], int size){

FILE \*file;

int i;

file = fopen("inventory.dat", "w+b");

// Open and erase the contents of the file

if (!file){

printf("Error while opening the file!\n");

return;

}

// Write the stucture to the file

for(i = 0;i < size;i++){

fwrite(&inventory[i], sizeof(Tproduct), 1, file);

}

// Close the data file

fclose(file);

return;

}

int binarySearch(Tproduct inventory[], int key, int size){

// Null array

if(null(size))

return -1;

if (!sortData){

// Sort products

sortProduct(inventory,size);

// Global variable

sortData = 1;

}

// Found the product

int start = 0,end = size, middle;

while (start <= end){

middle = (int)(start+end) / 2;

if(inventory[middle].productCode == key)

return middle;

if(inventory[middle].productCode < key)

start = middle+1;

else

end = middle-1;

}

// Did not find the product

return -1;

}

int null(int size){

// Null

if(size==0){

printf("\nEMPTY REGISTRY!\n");

return 1;

}

// Filled in

return 0;

}

void sortProduct(Tproduct inventory[], int size){

int i,j;

Tproduct aux;

for(i=0;i<size-1;i++)

for(j=i+1;j<size;j++)

if (inventory[i].productCode > inventory[j].productCode){

aux=inventory[i];

inventory[i]=inventory[j];

inventory[j]=aux;

}

}

void addProdutct(Tproduct inventory[], int\* size){

if(\*size == MAX){

printf("\nERROR!\nFILE IS FULL.\n");

return;

}

Tproduct aux;

char correct;

do{

printf("Code: ");

scanf("%ld", &aux.productCode);

cleanStdin();

} while (aux.productCode <= 0);

if(binarySearch(inventory, aux.productCode, \*size) >= 0){

printf("\nTHE PRODUCT CODE IS ALREDY REGISTERED !!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU\n");

getchar();

system("clear");

return;

}

do{

printf("\nProduct name: ");

fgets(aux.name, sizeof(aux.name), stdin);

cleanStdin();

aux.name[strcspn(aux.name, "\n")] = '\0';

if (aux.name[0] == '\0') {

printf("THE FIELD CANT BE EMPTY\n");

}

} while(aux.name[0] == '\0');

do{

printf("\nDescription of the product: ");

fgets(aux.productDescription, sizeof(aux.productDescription), stdin);

cleanStdin();

aux.productDescription[strcspn(aux.productDescription, "\n")] = '\0';

if (aux.productDescription[0] == '\0') {

printf("THE FIELD CANT BE EMPTY\n");

}

} while (aux.productDescription[0] == '\0');

do{

printf("\nSupplier of the product: ");

fgets(aux.productSupplier, sizeof(aux.productSupplier), stdin);

cleanStdin();

aux.productSupplier[strcspn(aux.productSupplier, "\n")] = '\0';

if (aux.productSupplier[0] == '\0') {

printf("THE FIELD CANT BE EMPTY\n");

}

} while (aux.productSupplier[0] == '\0');

do{

printf("\nMinimal quantity of the product in the inventory: ");

scanf("%d", &aux.inventoryMin);

cleanStdin();

} while (aux.inventoryMin < 0);

do{

printf("\nGroup: \n");

printf("1 - HAIR : \n");

printf("2 - NAIL : \n");

printf("3 - SKIN : \n");

printf("CHOOSE ONE OF THE GROUPS ABOVE: ");

scanf("%d", &aux.productGroup);

cleanStdin();

if (aux.productGroup != 1 && aux.productGroup != 2 && aux.productGroup != 3){

printf("\nInvalid product group. Type again.\n");

}

} while (aux.productGroup != 1 && aux.productGroup != 2 && aux.productGroup != 3);

do{

printf("\nProduct quantity in inventory: ");

scanf("%d", &aux.productQuantity);

cleanStdin();

} while (aux.productQuantity < 0);

do{

printf("\nProduct Unit (UN or LT): ");

fgets(aux.productUnit, sizeof(aux.productUnit), stdin);

cleanStdin();

aux.productUnit[strcspn(aux.productUnit, "\n")] = '\0';

if (strcmp(aux.productUnit, "UN") != 0 && strcmp(aux.productUnit, "LT") != 0) {

printf("Invalid product unit. Type again.\n");

}

} while (strcmp(aux.productUnit, "UN") != 0 && strcmp(aux.productUnit, "LT") != 0);

do{

printf("\nBuy price of the product from the supplier: R$ ");

scanf("%f", &aux.productBuyPrice);

cleanStdin();

} while (aux.productBuyPrice < 0);

do{

printf("\nMinimal profit margin for the product: R$ ");

scanf("%f", &aux.productMinimalProfit);

cleanStdin();

} while (aux.productMinimalProfit < 0);

do{

printf("\nSell price of the product: R$ ");

scanf("%f", &aux.productSellPrice);

cleanStdin();

} while (aux.productSellPrice < 0);

aux.expiredDate = getExpiredDate();

printf("\nThe data is correct? (Y/N)");

correct = getchar();

cleanStdin();

system("clear");

if(toupper(correct) == 'Y'){

// Makes the transfer to the array

inventory[\*size] = aux;

(\*size)++;

sortData = 0;

printf("\tTHE PRODUCT WAS ADDED!!\n");

}

printf("\nTYPE ENTER TO GO BACK TO THE MENU\n");

cleanStdin();

getchar();

system("clear");

return;

}

void generalReport(Tproduct inventory[], int size){

struct tm \*dateToday;

time\_t dayT;

time(&dayT);

dateToday = localtime(&dayT);

int i, counter = 0;

// Define the limit of pages being shown per page

int limit = 2;

// Number of the current

int page = 1;

// Calculates the number of pages

int totalPages = (size + limit - 1) / limit;

cleanStdin();

printf("\tInventory control - General report - %d/%d/%d\n", dateToday->tm\_mday, \

dateToday->tm\_mon + 1, dateToday->tm\_year + 1900);

printf("=================================================================================================\n");

sortProduct(inventory, size);

while (counter < size) {

// Exhibit the current products of the page

for(i = counter; i < size && i < counter + limit; i++){

printf(" Product Code\t\t\t\t\t Group\n");

printf("\t%ld\t\t\t\t\t\t%d\n", inventory[i].productCode, inventory[i].productGroup);

printf("Name: %s\t\t\t\tDescription: %s\n", inventory[i].name, inventory[i].productDescription);

printf("Unit: %s\t\tSupplier: %s\n", inventory[i].productUnit, inventory[i].productSupplier);

printf("Buy price: R$ %.2f\tSell price: R$ %.2f\tMinimal profit margin: R$ %.2f\n", \

inventory[i].productBuyPrice, \

inventory[i].productSellPrice, inventory[i].productMinimalProfit);

printf("Product quantity in inventory: %d\tMinimal product quantity for the inventory: %d\n", inventory[i].productQuantity, \

inventory[i].inventoryMin);

printf("Expiration date: %d/%d/%d\n", inventory[i].expiredDate.tm\_mday, \

inventory[i].expiredDate.tm\_mon, inventory[i].expiredDate.tm\_year);

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

}

printf("Page %d of %d\n", page, totalPages);

if (page < totalPages) {

printf("TYPE 'N' TO GO TO THE NEXT PAGE ");

}

if (page > 1) {

printf("OR TYPE 'P' TO GO TO PREVIOUS PAGE ");

}

// Clean the character of the pending line

printf("OR TYPE 'Q' TO QUIT THE REPORT\n");

char opt = getchar();

getchar();

// Exits the loop if the user decides to go back to the menu

if (toupper(opt) == 'Q'){

system("clear");

break;

// Go to the next page

} else if (toupper(opt) == 'N') {

if (page < totalPages) {

counter += limit;

// Updates the number of the page

page++;

}

// Go back a page

} else if (toupper(opt) == 'P'){

if (page > 1) {

counter -= limit;

if (counter < 0) {

// Garantees that the counter cant be negative

counter = 0;

}

// Updates the page number

page--;

}

}

system("clear");

}

}

void reportPriceList(Tproduct inventory[], int size){

if (size == 0) {

printf("\tEMPTY INVENTORY!!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

struct tm \*dateToday;

time\_t dayT;

time(&dayT);

dateToday = localtime(&dayT);

int i;

int page = 1;

int totalPages = (size + 14) / 15;

cleanStdin();

printf("\tInventory control - Price list - %d/%d/%d\n", dateToday->tm\_mday, \

dateToday->tm\_mon + 1, dateToday->tm\_year + 1900);

printf("==========================================================================\n");

printf("\tCode\tName\t\t\t\t\t\tPrice\n");

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

for (i = 0; i < size; i++) {

printf("\t%ld\t%-45s R$ %.2f\n", \

inventory[i].productCode, inventory[i].name, inventory[i].productSellPrice);

if ((i + 1) % 15 == 0 && i + 1 < size) {

// If has more than 15 products to display, it waits the use action

printf("\nTYPE ANY KEY TO CONTINUE\n");

getchar();

system("clear");

printf("\tInventory control - Price list - %d/%d/%d\n", dateToday->tm\_mday, \

dateToday->tm\_mon + 1, dateToday->tm\_year + 1900);

printf("==========================================================================\n");

printf("Code Name Price\n");

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

page++;

}

}

printf("\n==========================================================================\n");

printf("\nPage %d of %d\n", page, totalPages);

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

getchar();

system("clear");

return;

}

void changeProduct(Tproduct inventory[], int size){

if (null(size)) {

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

long int productCode;

int position, opt;

printf("Type the code of the product were is gonna be changed: ");

scanf("%ld", &productCode);

position = binarySearch(inventory, productCode, size);

if (position == -1) {

printf("PRODUCT NOT FOUND!!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

printf("======================================================\n");

printf(" 1 - NAME\n");

printf(" 2 - GROUP\n");

printf(" 3 - DESCRIPTION\n");

printf(" 4 - UNIT\n");

printf(" 5 - SUPPLIER\n");

printf(" 6 - PRODUCT QUANTITY\n");

printf(" 7 - BUY PRICE\n");

printf(" 8 - SELL PRICE\n");

printf(" 9 - MINIMAL PROFIT MARGIN\n");

printf(" 10 - MINIMAL PRODUCT QUANTITY FOR THE INVENTORY\n");

//printf(" 11 - EXPIRATION DATE\n");

//printf(" 12 - ALL THE OPTIONS ABOVE\n");

printf(" 0 - EXIT\n");

printf("\nCHOOSE THE DESIRED OPTION TO EDIT:\n");

scanf("%d",&opt);

system("clear");

switch(opt){

case 1:

printf("Type the new name of the product: ");

cleanStdin();

fgets(inventory[position].name, 41, stdin);

inventory[position].name[strcspn(inventory[position].name, "\n")] = '\0';

break;

case 2:

printf("Type the new group that the product belongs: ");

scanf("%d", &inventory[position].productGroup);

break;

case 3:

printf("Type the new description of the product: ");

cleanStdin();

fgets(inventory[position].productDescription, 41, stdin);

inventory[position].productDescription[strcspn(inventory[position].productDescription, "\n")] = '\0';

break;

case 4:

printf("Type the new unit: ");

cleanStdin();

fgets(inventory[position].productUnit, 3, stdin);

inventory[position].productUnit[strcspn(inventory[position].productUnit, "\n")] = '\0';

break;

case 5:

printf("Type the new name of the supplier of the product: ");

cleanStdin();

fgets(inventory[position].productSupplier, 41, stdin);

inventory[position].productSupplier[strcspn(inventory[position].productSupplier, "\n")] = '\0';

break;

case 6:

printf("Type the new quantity of the product in the inventory: ");

scanf("%d", &inventory[position].productQuantity);

break;

case 7:

printf("Type the new buy price of the product: ");

scanf("%f", &inventory[position].productBuyPrice);

break;

case 8:

printf("Type the new sell price of the product: ");

scanf("%f", &inventory[position].productSellPrice);

break;

case 9:

printf("Type the new minimal profit margin of the product: ");

scanf("%f", &inventory[position].productMinimalProfit);

break;

case 10:

printf("Type the new minimal quantity of the product necessary to be in the inventory: ");

scanf("%d", &inventory[position].inventoryMin);

break;

/\*case 11:

changeExpiredMenu(inventory, size);

break;

case 11:

changeFunction(inventory, size);

break;\*/

case 0:

return;

break;

default:

printf("INVALID OPTION!!\n\n");

system("clear");

}

printf("\nPRODUCT MODIFIED WITH SUCCESS!!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

void deleteProduct(Tproduct inventory[], int\* size) {

if (null(\*size)) {

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

long int productCode;

int position, i;

printf("Type the code of the product desired to be deleted: ");

scanf("%ld", &productCode);

position = binarySearch(inventory, productCode, \*size);

if (position == -1) {

printf("PRODUCT NOT FOUND!!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

for (i = position; i < (\*size) - 1; i++) {

inventory[i] = inventory[i + 1];

}

(\*size)--;

sortData = 0;

printf("\n PRODUCT DELETED WITH SUCCESS!!... \n ");

system("\n pause \n");

system("clear");

}

void consultProduct(Tproduct inventory[], int size){

if (null(size)) {

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

long int productCode;

int position;

printf("Type the code of the product desired to be consulted: ");

scanf("%ld", &productCode);

position = binarySearch(inventory, productCode, size);

if(position == -1){

printf("PRODUCT NOT FOUND!!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

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

printf("Code: %ld\n", inventory[position].productCode);

printf("Name: %s", inventory[position].name);

printf("Group: %d\n", inventory[position].productGroup);

printf("Description: %s\n", inventory[position].productDescription);

printf("Unit: %s\n", inventory[position].productUnit);

printf("Supplier: %s\n", inventory[position].productSupplier);

printf("Quantity in inventory: %.d\n", inventory[position].productQuantity);

printf("Buy Price: R$ %.2f\n", inventory[position].productBuyPrice);

printf("Sell Price: R$ %.2f\n", inventory[position].productSellPrice);

printf("Minimal profit margin: R$ %.2f\n", inventory[position].productMinimalProfit);

printf("Minimal product quantity in inventory: %d\n", inventory[position].inventoryMin);

printf("Expiration date: %d/%d/%d\n", inventory[position].expiredDate.tm\_mday, \

inventory[position].expiredDate.tm\_mon, inventory[position].expiredDate.tm\_year);

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

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

int consultProductByName(Tproduct inventory[], int size) {

char name[41];

int productsFound = 0;

printf("Type the name of the searched product: ");

cleanStdin();

fgets(name, sizeof(name), stdin);

name[strcspn(name, "\n")] = '\0';

printf("Products found with the name '%s':\n", name);

int i;

for (i = 0; i < size; i++) {

if (strcmp(inventory[i].name, name) == 0) {

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

printf("Code: %ld\n", inventory[i].productCode);

printf("Name: %s\n", inventory[i].name);

printf("Group: %d\n", inventory[i].productGroup);

printf("Description: %s\n", inventory[i].productDescription);

printf("Unit: %s\n", inventory[i].productUnit);

printf("Supplier: %s\n", inventory[i].productSupplier);

printf("Quantity in inventory: %.d\n", inventory[i].productQuantity);

printf("Buy Price: R$ %.2f\n", inventory[i].productBuyPrice);

printf("Sell Price: R$ %.2f\n", inventory[i].productSellPrice);

printf("Minimal profit margin: R$ %.2f\n", inventory[i].productMinimalProfit);

printf("Minimal product quantity in inventory: %d\n", inventory[i].inventoryMin);

printf("Expiration date: %d/%d/%d\n", inventory[i].expiredDate.tm\_mday, \

inventory[i].expiredDate.tm\_mon, inventory[i].expiredDate.tm\_year);

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

productsFound++;

}

}

if (productsFound == 0) {

printf("\nPRODUCT NOT FOUND!\n");

}

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return productsFound;

}

void reportMinimalInventory(Tproduct inventory[], int size){

if (size == 0) {

printf("\tEMPTY INVENTORY!!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

int i;

int productFound = 0;

for (i = 0; i < size; i++) {

if (inventory[i].productQuantity < inventory[i].inventoryMin) {

if (productFound == 0) {

cleanStdin();

printf("There are products bellow the minimal quantity!\n\n");

}

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

printf("Code: %ld \n", inventory[i].productCode);

printf("Name: %s \n", inventory[i].name);

printf("Group: %d\n", inventory[i].productGroup);

printf("Description: %s\n", inventory[i].productDescription);

printf("Unit: %s\n", inventory[i].productUnit);

printf("Supplier: %s\n", inventory[i].productSupplier);

printf("Quantity in inventory: %d\n", inventory[i].productQuantity);

printf("Buy price: R$ %.2f\n", inventory[i].productBuyPrice);

printf("Sell price: R$ %.2f\n", inventory[i].productSellPrice);

printf("Minimal margin profit: R$ %.2f\n", inventory[i].productMinimalProfit);

printf("Minimal quantity in inventory: %d\n", inventory[i].inventoryMin);

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

printf("\n");

productFound++;

}

}

if (productFound == 0) {

cleanStdin();

printf("No product is bellow the minimal quantity in the inventory!\n\n");

}

printf("TYPE ENTER TO GO BACK TO THE MENU");

getchar();

system("clear");

return;

}

void increaseSupplierPrices(Tproduct inventory[], int size){

char productSupplier[50];

float percentageAdjustment;

printf("Inform supplier name: ");

cleanStdin();

fgets(productSupplier, 50, stdin);

// Remove the charactere of the break line

cleanStdin();

productSupplier[strcspn(productSupplier, "\n")] = '\0';

printf("Inform the percentage adjustment in the buy price: ");

scanf("%f", &percentageAdjustment);

int i;

int alteredProducts = 0;

for(i = 0; i < size; i++){

if(strcmp(inventory[i].productSupplier, productSupplier) == 0){

printf("\nProduct found:\n");

printf("Code: %ld\n", inventory[i].productCode);

printf("Name: %s", inventory[i].name);

printf("Description: %s\n", inventory[i].productDescription);

printf("Current product buy price: R$ %.2f\n", inventory[i].productBuyPrice);

float newProductBuyPrice = inventory[i].productBuyPrice \* (1 + percentageAdjustment / 100);

printf("New product buy price: R$ %.2f\n", newProductBuyPrice);

printf("Type 'Y' to confirm the change: ");

char opt;

cleanStdin();

scanf("%c", &opt);

if(toupper(opt) == 'Y'){

inventory[i].productBuyPrice = newProductBuyPrice;

alteredProducts++;

printf("Increase in the buy price of the product with the code %ld.\n", \

inventory[i].productCode);

}

}

}

if(alteredProducts > 0){

printf("\nPrice increase of the buy price of the supplier products completed. \

Total of products were prices were increased: %d\n",

alteredProducts);

} else{

printf("\nNo product was found for the informed supplier.\n");

}

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

void espMenu(Tproduct inventory[], int size){

struct tm \*date;

time\_t dayT;

time(&dayT);

date = localtime(&dayT);

int opt;

do{

printf("Today is: %d/%d/%d\n", date->tm\_mday, date->tm\_mon + 1, date->tm\_year + 1900);

printf("\n===== SPECIAL REPORTS =====\n");

printf("1 - REPORTS OF PRODUCTS BELLOW THE MINIMAL QUANTITY : \n");

printf("2 - INCREASE THE BUY PRICE OF PRODUCTS FROM A SUPPLIER\n");

printf("3 - REPORT WITH EXPIRED PRODUCTS (TESTING - NOT WORKING CORRECTLY YET)\n");

printf("0 - GO BACK TO MAIN MENU\n");

printf("Choose an options: ");

scanf("%d", &opt);

system("clear");

switch (opt){

case 1:

reportMinimalInventory(inventory, size);

break;

case 2:

increaseSupplierPrices(inventory, size);

writeFile(inventory, size);

break;

case 3:

reportExpiredProduct(inventory, size);

break;

case 0:

return;

break;

default:

printf("INVALID OPTION!!\n\n");

system("clear");

}

} while (opt != 0);

}

/\*

void changeExpiredMenu(Tproduct inventory[], int size){

int opt, position;

printf("Type the option to edit the expired date:\n");

printf("1 - Day ");

printf("2 - Month");

printf("3 - Year");

printf("4 - The 3 choises");

scanf("%d", & opt);

switch (opt){

case 1:

printf("Type: ");

scanf("%d", &inventory[position].expiredDate.tm\_mday);

break;

case 2:

printf("Type the new group that the product belongs: ");

scanf("%d", &inventory[position].expiredDate.tm\_mon);

break;

case 3:

printf("Type the new group that the product belongs: ");

scanf("%d", &inventory[position].expiredDate.tm\_year);

break;

case 4:

printf("Type the new group that the product belongs: ");

scanf("%d", &inventory[position].expiredDate.tm\_mday);

printf("Type the new group that the product belongs: ");

scanf("%d", &inventory[position].expiredDate.tm\_mon);

printf("Type the new group that the product belongs: ");

scanf("%d", &inventory[position].expiredDate.tm\_year);

cleanStdin();

break;

default:

printf("INVALID OPTION!!\n\n");

}

}\*/

void listReport(Tproduct inventory[], int index){

printf("%ld\t\t\t\t%.2f\n", inventory[index].productCode, \

inventory[index].productSellPrice);

}

/\*void changeFunction(Tproduct inventory[], int size){

int position;

printf("Type the new group that the product belongs: ");

scanf("%d", &inventory[position].productGroup);

printf("Type the new name of the product: ");

cleanStdin();

fgets(inventory[position].name, 41, stdin);

inventory[position].name[strcspn(inventory[position].name, "\n")] = '\0';

printf("Type the new description of the product: ");

cleanStdin();

fgets(inventory[position].productDescription, 41, stdin);

inventory[position].productDescription[strcspn(inventory[position].productDescription, "\n")] = '\0';

printf("Type the new unit: ");

cleanStdin();

fgets(inventory[position].productUnit, 3, stdin);

inventory[position].productUnit[strcspn(inventory[position].productUnit, "\n")] = '\0';

printf("Type the new name of the supplier of the product: ");

cleanStdin();

fgets(inventory[position].productSupplier, 41, stdin);

inventory[position].productSupplier[strcspn(inventory[position].productSupplier, "\n")] = '\0';

printf("Type the new quantity of the product in the inventory: ");

scanf("%d", &inventory[position].productQuantity);

printf("Type the new buy price of the product: ");

scanf("%f", &inventory[position].productBuyPrice);

printf("Type the new sell price of the product: ");

scanf("%f", &inventory[position].productSellPrice);

printf("Type the new minimal profit margin of the product: ");

scanf("%f", &inventory[position].productMinimalProfit);

printf("Type the new minimal quantity of the product necessary to be in the inventory: ");

scanf("%d", &inventory[position].inventoryMin);

changeExpiredMenu(inventory, size);

}\*/

void reportExpiredProduct(Tproduct inventory[], int size){

struct tm \*date;

time\_t dayT;

time(&dayT);

date = localtime(&dayT);

if (size == 0) {

printf("\tEMPTY INVENTORY!!\n");

printf("\nTYPE ENTER TO GO BACK TO THE MENU");

cleanStdin();

getchar();

system("clear");

return;

}

int i;

int productFound = 0;

// This part has a little problem were in previous months, \

if number of the day is bigger than the current day, WON

for (i = 0; i < size; i++) {

if(inventory[i].expiredDate.tm\_year <= date->tm\_year + 1900 \

&& inventory[i].expiredDate.tm\_mon <= date->tm\_mon + 1 \

&& inventory[i].expiredDate.tm\_mday <= date->tm\_mday \

|| inventory[i].expiredDate.tm\_year <= date->tm\_year + 1900 \

&& inventory[i].expiredDate.tm\_mon < date->tm\_mon + 1 \

&& inventory[i].expiredDate.tm\_mday >= date->tm\_mday){

if (productFound == 0) {

cleanStdin();

printf("There are products that expired!!\n\n");

}

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

printf("Code: %ld \n", inventory[i].productCode);

printf("Group: %d\n", inventory[i].productGroup);

printf("Description: %s\n", inventory[i].productDescription);

printf("Unit: %s\n", inventory[i].productUnit);

printf("Supplier: %s\n", inventory[i].productSupplier);

printf("Quantity in inventory: %d\n", inventory[i].productQuantity);

printf("Buy price: %.2f\n", inventory[i].productBuyPrice);

printf("Sell price: %.2f\n", inventory[i].productSellPrice);

printf("Minimal margin profit: %.2f\n", inventory[i].productMinimalProfit);

printf("Minimal quantity in inventory: %d\n", inventory[i].inventoryMin);

printf("Expiration date: %d/%d/%d\n", inventory[i].expiredDate.tm\_mday, \

inventory[i].expiredDate.tm\_mon, inventory[i].expiredDate.tm\_year);

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

printf("\n");

productFound++;

}

}

if (productFound == 0) {

cleanStdin();

printf("No product is expired!\n\n");

}

printf("TYPE ENTER TO GO BACK TO THE MENU");

getchar();

system("clear");

return;

}

void cleanStdin(void){

int c;

do {

c = getchar();

} while (c != '\n' && c != EOF);

}

int dateExp(struct tm dateE){

// Checks if is a valid date

if (dateE.tm\_year < 1900){

printf("INVALID DATE\n");

return 0;

}

if (dateE.tm\_mon < 1 || dateE.tm\_mon > 12){

printf("INVALID DATE\n!!");

return 0;

}

if (dateE.tm\_mday < 1 || dateE.tm\_mday > 31){

printf("INVALID DATE!!\n");

return 0;

}

return 1;

}

## CONSIDERAÇÕES FINAIS

Com esse programa, é possível monitorar os produtos de um salão de beleza, resultando em um melhor gerenciamento do estoque.

Primeiramente, a fiscalização do sistema fez-se necessária, pois sem a devida organização dos dados podem ocasionar em acidentes que prejudiquem a saúde do cliente e abalar a reputação do estabelecimento.

Ademais, ter um sistema que te notifique caso haja um problema com o produto a ser vendido ou utilizado indica ser um dos melhores métodos para um trabalho harmônico. Portanto, possuir um sistema de controle de estoque é essencial para o funcionamento de um salão de beleza.

## BIBLIOGRAFIA E LINKS

Repositório: https://github.com/I7sReact/controle-de-estoque