ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Projeto de Laboratório de Programação

Licenciatura em Engenharia Informática

2022/2023

Grupo 119

Francisco Costa – 8220805

Eduardo Ferreira – 8220823

Miguel Coimbra – 8220844

ÍNDICE

1 Introdução

1.1 Propósito

Este documento é apresentado com o âmbito de descrever as especificações de requisitos para desenvolver uma aplicação para a empresa "Móveis para Todos" proposta com a ESTG. A avaliação e exposição de todos os requisitos estão descritos ao longo do documento de uma forma sistemática. Esta exposição está direcionada a membros da empresa, desenvolvedores ou a todas as partes interessadas no processo.

1.2 Âmbito do Projeto

O objetivo desta aplicação é oferecer á "Móveis para Todos" um sistema transversal a toda a organização para a gestão dos seus processos internos, registo de dados vitais ao funcionamento da empresa e interação entre o cliente e a empresa. Com o desenvolvimento desta solução pretende-se um aumento da informatização e digitalização dos procedimentos da empresa de modo a garantir uma maior eficácia dos processos.

Caracterizando de forma mais concreta, o software descrito é capaz de suportar, de forma automática e otimizada, todo o processo de compra, venda e registo de móveis. Deve também ser apto a interação com o cliente direto facilitando o processo de compra dos produtos expostos. Por outro lado, também deve ser capaz de facilitar a gestão por parte de um *Admin*, gerindo dados da correlação entre as encomendas feitas e os produtos existentes como também gerir informações sobre os seus compradores.

1.3 **Caracterização do Programa**

A "Móveis para Todos", como descrito anteriormente, é uma empresa de móveis, responsável pelo fabrico e venda destes mesmos produtos a um cliente final. O programa a ser desenvolvido deve ter todo um sistema e interface de interação com o cliente informatizados, recorrendo a uma formatação CRUD (conceito explicado posteriormente), de gestão por parte de um perfil *Admin.*

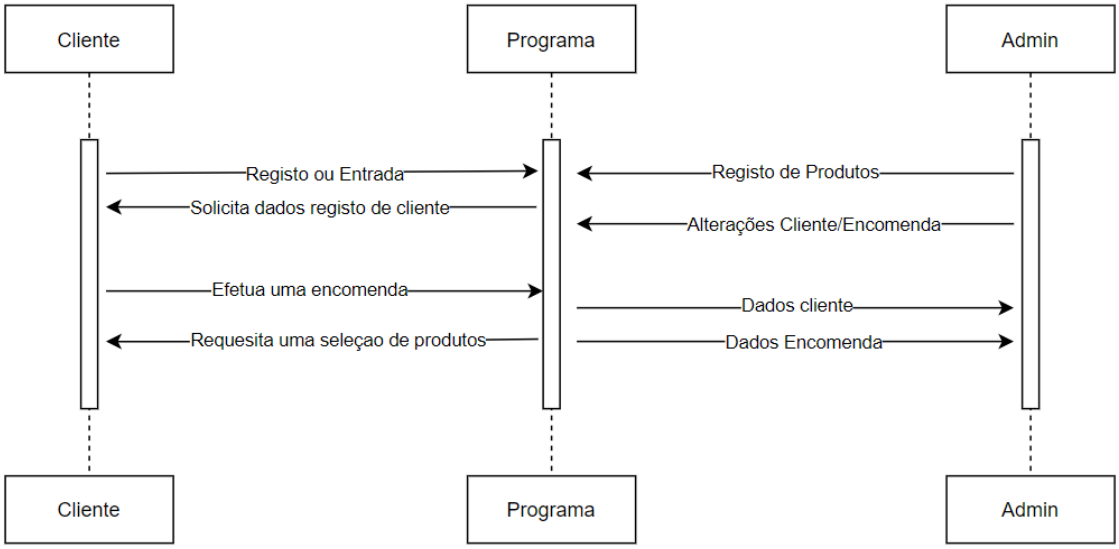

Figura 1. Diagrama de Sistema para descrever interações no sistema Cliente/Programa/Admin na "Móveis para Todos".

Para além de informatizar a sequência de ações, deve ser também capaz de criar outputs de informação. Por exemplo, criar listas de componentes essenciais a dar resposta a determinadas encomendas de uma semana.

Avaliando a organização interna da empresa "Móveis para Todos" é importante ter também em consideração que a empresa se responsabiliza pela criação, fabrico e venda de produtos, posto isto é essencial que exista uma troca de informação, Cliente/Máquina e Admin/Máquina (Figura 1.), prática e eficaz.

1.4 **Planificação inicial**

Com uma leitura pormenorizada do relatório de objetivos e especificações do projeto, fornecidas em documento ao grupo, iniciamos a planificação do projeto em formato esquema (Figura 2.), em que de uma forma breve criamos um esboço do programa requerido e dos caminhos a percorrer para que este fosse funcional e respondesse a todas as exigências descritas no documento.

Dada uma breve discussão é proposto um ponto de partida com os menus, mas com o tempo percebemos que estes seriam só uma fase posterior do projeto, e que seguindo este método iria dificultar a progressão implicando a que, a cada implementação funcional, uma verificação lógica seria necessária.

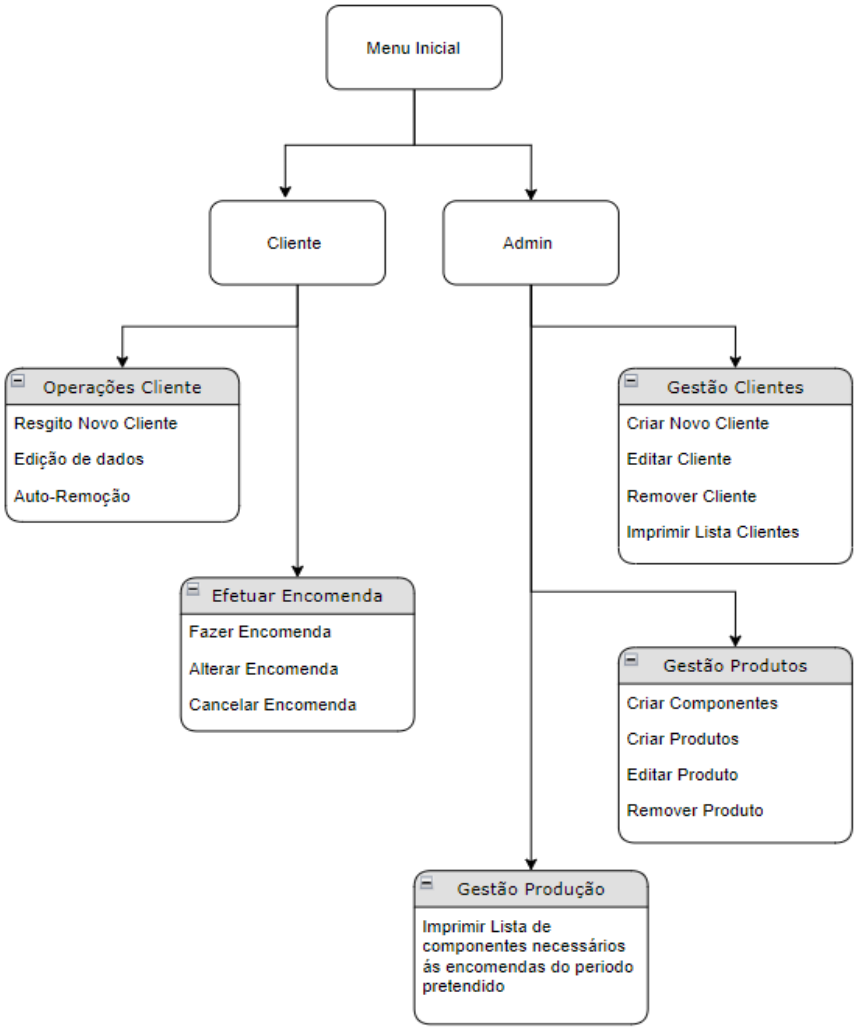

Figura 2. Esquema de planificação inicial do projeto.

2. Funcionalidades Requeridas

2.1 Tipos de Utilizador e as suas características

Tal como evidente na secção anterior, irá ser necessário criar vários tipos de utilizadores, sendo importante defini-los e identificar as suas hierarquias de modo a facilitar a atribuição de permissões no próprio sistema.

Através da análise da estruturação da empresa, percebemos que é necessária a criação de um cargo *Administrador* com funções relacionadas á gestão, e um cargo de cliente com capacidades de compra e registo. Obstantes de que apenas existem dois tipos de utilizadores nas funcionalidades requeridas, decidimos enquanto grupo que uma das funcionalidades propostas seria a criação de um utilizador, "Funcionário" com habilitações que permitissem uma maior prática no seu trabalho quotidiano.

2.1.1 Utilizador - "Administrador"

O cargo de administrador é hierarquicamente o cargo com mais permissões no programa com capacidades de gestão e manipulação de dados, este administrador é capaz de usar as funções como Gestão de clientes, produtos e produção.

2.1.2 Utilizador – "Cliente"

No utilizador cliente, estão presentes funções como registo, alteração de dados pessoais e auto remoção todas estas ligadas ao registo numa lista, passível de ser consultada pelo administrador. Pode também efetuar encomendas, alterá-las e cancelar se assim o desejar.

2.2 Funcionalidades Obrigatórias

Nesta secção serão introduzidas as funções com presença obrigatória no projeto tal como descrito no documento entregue ao grupo.

2.2.1 Metodologia de Trabalho

Posteriormente á planificação inicial (Figura 2.), o grupo entendeu que seria adequada uma estrutura de trabalho do tipo CRUD. Esta metodologia aplicada consiste numa divisão da estrutura em quatro partes. Para que exista um input de dados é inicialmente criada uma função de registo *"C-Create"*, para que estes sejam utilizados pelo programa, é consequentemente necessária uma função de leitura de dados *"R-Read"*, no caso de os dados entrarem em desuso ou necessitarem de uma atualização serão implementadas mais duas funções, de atualização *"U-Update"* ou eliminação *"D-Delete"*.

Dado existirem variáveis que precisam armazenar mais do que um valor ou diferentes tipos de dados, foi criada uma biblioteca com *Structs* onde para um tipo de dados são agrupadas diferentes variáveis (Figura 3.), caso este utilizado inúmeras vezes. Todas estas structs são iniciadas em main.c.

```
typedef struct{
    int id;
    int nif;
    char nome[MAX];
    char morada[MAX];
    char pais[MAX];
    int nencomendas;
    char atividade[7];
}Cliente;
```

Figura 3. Exemplo de struct implementada.

2.3 Funções Implementadas

Como descrito no subcapítulo anterior seria necessária uma estrutura capaz de receber dados de um utilizador, efetuar a leitura dos mesmos e também a sua manipulação utilizando a estrutura CRUD.

2.3.1 Administrador

2.3.1.1 Gestão de Clientes

O conjunto de funções Gerir Cliente é apenas de manipulação possível ao administrador, cargo este que acarreta as maiores permissões da empresa. Esta função é dada como opção no menu de administrador.

Registo de Cliente

Esta função permite ao utilizador o registo de clientes (Figura 4.), é inicialmente necessária a introdução dos dados relativos ao cliente, nestes dados consta um ID que será comparado aos já existentes na lista de clientes, para que se for verificado este não será registado, evitando clientes repetidos.

```
case 1://regista clientes , verifica se o registou
    registarCliente(&cliente);
    int inseriu = inserirCliente(&loja, &cliente);
    if (inseriu == 0) {
        printf("Cliente com este id ja existe nao foi adicionado\n");
    } else {
        printf("Cliente adicionado com sucesso\n");
    }
    c = obterCaratere(CARATERE);
    limpaBuffer();
    break;
```

Figura 4. Registo de Clientes

Edição de Cliente

Com esta funcionalidade é possível ao administrador alterar os dados de um cliente já presente na lista de clientes, sendo comprovada a sua existência inicialmente. Os dados são alterados individualmente a cada iteração, para que não seja necessária uma repetição de dados que não se pretende alterar.

```
case 2://edicao de clientes
    imprimirTudo(&loja);
    printf("Qual o id do cliente que deseja atualizar!?\n-\n");
    scanf("%d", &idprocura);
    limpaBuffer();
    printf("O que deseja atualizar no cliente:\n1-nif\n2-nome\n3-pais\n4-morada\n-");
    scanf("%d", &opu);
    limpaBuffer();
    switch (opu) {
        case 1:
            printf("\nDigite o novo nif\n");
            scanf("%d", &nifnovo);
            limpaBuffer();
            atualizarClienteNif(&loja, &idprocura, nifnovo);
            c = obterCaratere(CARATERE);
            break;
        case 2:
            printf("\nDigite o novo nome\n");
            scanf("%s", &nomenovo);
            limpaBuffer();
            atualizarClienteNome(&loja, &idprocura, nomenovo);
            c = obterCaratere(CARATERE);
            break;
        case 3:
            printf("\nDigite o novo pais\n");
            scanf("%s", &paisnovo);
            limpaBuffer();
            atualizarClientePais(&loja, &idprocura, paisnovo);
            c = obterCaratere(CARATERE);
            break;
        case 4:
            printf("\nDigite a nova morada\n");
            scanf("%s", &moradanova);
            limpaBuffer();
            atualizarClienteMorada(&loja, &idprocura, moradanova);
            c = obterCaratere(CARATERE);
            break;
    }
```

Figura 5. Função de edição de clientes

Remover Clientes

Caso um cliente não tenha efetuado encomendas desde o seu registo é possível com esta funcionalidade a remoção do cliente (Figura 6.). A um cliente que tenha efetuado encomendas anteriormente e não o faça num longo período é possível definir o seu estado para inativo.

```
case 3://remove os clientes
    imprimirTudo(&loja);
    printf("\n");
    printf("Qual o id do cliente que deseja eliminar!?");
    scanf("%d", &idprocura);
    limpaBuffer();
    removerCliente(&loja, &idprocura);
    c = obterCaratere(CARATERE);


    break;
```

Figura 6. Função para Remover Cliente

Listar Clientes

Para uma noção dos clientes atuais, é possível ao administrador requerer uma listagem (Figura 7.)

```
case 4://lista todos os clientes adicionados
    imprimirTudo(&loja);
    c = obterCaratere(CARATERE);

    break;
```

Figura 7. Função para Listagem de Clientes

2.3.1.2 Gestão de Produtos

O conjunto de funções de gestão de produtos é também apenas possível ao administrador, no entanto ao contrário dos clientes, nos produtos são utilizadas duas structs para guardar variáveis, no caso materiais e produtos.

Materiais

Registo de Materiais

A gestão de produtos é iniciada com a inserção de materiais (Figura 8.), que serão dispostos numa lista, para futuro uso na criação de produtos com a associação de materiais.

```
case 1://registar materiais
    registarComponente(&componente);
    int inseriuc = inserirComponente(&armazem, &componente);
    if (inseriuc == 0) {
        printf("Componente com este id ja existe nao foi adicionado\n");
    } else {
        printf("Componente adicionado com sucesso\n");
    }
    c = obterCaratere(CARATERE);
    limpaBuffer();
    break;
```

Figura 8. Função para registo de Materiais

Edição de Materiais

No caso de ser necessária a atualização dos dados referentes a materiais previamente inseridos, surge a função de edição de materiais (Figura 9.), que permite a edição das suas componentes em separado evitando uma necessidade de inserir dados repetidos.

```
case 2://editar materiais
    imprimirTodosComponentes(&armazem);
    printf("Qual o codigo do componente da lista a cima que deseja atualizar!?\n-\n");
    scanf("%s", codmat);
    limpaBuffer();
    printf("O que deseja atualizar no componente:\n1-descricao\n2-unidade\n-");
    scanf("%d", &opu);
    limpaBuffer();
    switch (opu) {
        case 1:
            printf("\nDigite a nova descricao\n");
            scanf("%s", descricaonova);
            limpaBuffer();
            atualizarComponenteDescricao(&armazem, codmat, descricaonova);
        case 2:
            printf("\nDigite se Unidade -UN se par -PAR\n");
            scanf("%s", unidadenova);
            limpaBuffer();
            atualizarComponenteUnidade(&armazem, codmat, unidadenova);
    }


    break;
```

Figura 9. Função para edição de Materiais

Remoção e Listagem de Materiais

Tal como nos clientes são necessárias funções de *display* e *remove* (Figura 10.) para o tratamento de dados por parte do administrado processando-se da mesma forma da situação descrita anteriormente.

```
case 3://remover materiais
    imprimirTodosComponentes(&armazem);
    printf("Qual o codigo do componente da lista a cima que deseja remover!?\n-\n");
    scanf("%s", codmat);
    limpaBuffer();
    removerComponente(&armazem, codmat);
    c = obterCaratere(CARATERE);
    limpaBuffer();
    break;

case 4://listar materiais
    imprimirTodosComponentes(&armazem);
    c = obterCaratere(CARATERE);
    limpaBuffer();
    break;
```

Figura 10. Funções de Remoção e Listagem de Materiais

Produtos

Registo e Edição de Produtos

Como este tipo de função é recorrente ao longo do projeto as funções de registo e edição utilizam a mesma estrutura usada nos subcapítulos anteriores, modificando-se apenas o tipo de variáveis e dados necessário, traduzido e exportado.

Remoção e Listagem de Produtos

A remoção e listagem de produtos foi também utilizada na secção dos materiais levando a que a função criada seja idêntica e transversal a todas as implementações.

2.3.1.3 Gestão de Produção

Visando um output em formato lista dos componentes necessário a responder às encomendas de um intervalo de tempo, a função de gestão de produção permite ao utilizador a introdução de um intervalo de tempo desejado (Figura 11.), na qual conseguirá consultar os componentes necessários para o fabrico dos produtos que constarem nas encomendas ativas. Esta função requereu por parte do grupo uma implementação de grande complexidade graças a utilização funções presentes em librarias pouco exploradas pelos membros do grupo, no entanto é considerada uma experiência enriquecedora no âmbito da aprendizagem.

Introdução do intervalo desejado

```
void gestaoProducao(Encomendas * encomendas, Produtos *produtos, Componentes *componentes) {

    int i, j, k, d, f;
    Data dataEscolhida ;
    printf("\nDigite o dia em que a semana começa\n");
    scanf("%d",&dataEscolhida.dia);
    printf("\nDigite o mes em que a semana começa\n");
    scanf("%d",&dataEscolhida.mes);
    printf("\nDigite o ano em que a semana começa\n");
    scanf("%d",&dataEscolhida.ano);
    struct tm novaData = {.tm_mday = dataEscolhida.dia, .tm_mon = dataEscolhida.mes, .tm_year = dataEscolhida.ano};
    novaData.tm_mday += 7;
    mktime(&novaData);
```

Figura 11. Componente relativa ao intervalo de tempo

Algoritmo de consulta de encomendas e listagem

Para ser possível uma consulta das encomendas no intervalo de tempo desejado foi necessária a aplicação de funções presentes em bibliotecas do predefinidas no IDE relativas a manipulação de variáveis temporais como é percetível na representação (Figura 12.).

```
for (i = 0; i < encomendas->contadore; i++) {
    if ((encomendas->listaEncomendas[i].dataentrega.dia >= dataEscolhida.dia && encomendas->listaEncomendas[i].dataentrega.dia <= novaData.tm_mday &&
         encomendas->listaEncomendas[i].dataentrega.mes >= dataEscolhida.mes && encomendas->listaEncomendas[i].dataentrega.mes <= novaData.tm_mon &&
         encomendas->listaEncomendas[i].dataentrega.ano >= dataEscolhida.ano && encomendas->listaEncomendas[i].dataentrega.ano <= novaData.tm_year)) {
        for (j = 0; j < encomendas->listaEncomendas[i].contadorprode; j++) {
            for (k = 0; k < produtos->contadorp; k++) {
                if (strcmp(produtos->listaProdutos[i].codProduto, encomendas->listaEncomendas[i].prode[j].cod) == 0) {
                    for (d = 0; d < produtos->listaProdutos[k].contadorcomp; d++) {
                        for (f = 0; f < componentes->contador; f++) {
                            if (strcmp(componentes->listacomponente[f].codMaterial, produtos->listaProdutos[k].composicao[f].codigo) == 0) {
                                printf("codigo: %s\n", componentes->listacomponente[f].codMaterial);
                                printf("descricao : %s\n", componentes->listacomponente[f].descricao);
                                printf("unidade:%s\n", componentes->listacomponente[f].unidade);
                                printf("Quantidade :%d\n", encomendas->listaEncomendas[d].prode[d].quantidadeprod * produtos->listaProdutos[f].composicao[j].quantidade);
                            }
                        }
                    }
                }
            }
        }
    }
```

Figura 12 Função responsável pela consulta de encomendas

2.3.2 Cliente

No menu de cliente será possível interações relativas a registo e dados pessoais, mas também efetuar encomendas, consultar o seu estado ou cancelar as mesmas.

2.3.2.1 Registo de Clientes

Para a efetivação de uma encomenda serão necessários dados pessoais de um cliente, posto isto é necessário criar uma função de registo de clientes, através de, como já criado anteriormente para produtos e materiais, uma lista de clientes onde constarão do tipo cliente (*struct* que alberga os dados relativos ao cliente), inúmeros dados relativos a cada registo (Figura 13.).

```
void registarCliente(Cliente *cliente) {
    printf("Digite o seu id:\n-");
    scanf("%d", &cliente->id);
    limpaBuffer();
    printf("Digite o seu nif:\n-");
    scanf("%d", &cliente->nif);
    limpaBuffer();
    printf("Digite o seu nome:\n-");
    scanf("%s", &cliente->nome);
    limpaBuffer();
    printf("Digite o seu morada:\n-");
    scanf("%s", &cliente->morada);
    limpaBuffer();
    printf("Digite o seu pais:\n-");
    scanf("%s", &cliente->pais);
    limpaBuffer();
    cliente->nencomendas = 0;
    strcpy(cliente->atividade, "Ativo");
}
```

Figura 13. Função para registo de cliente.

2.3.2.2 Edição de dados pessoais

O cliente pode também alterar os seus dados após o registo, através de uma simples estrutura de substituição de dados previamente utilizada no utilizador do administrador para produtos e Materiais. Esta estrutura possibilita a que seja possível a modificação de dados e a permanência dos anteriores em caso de não serem alterados (Figura.14&15).

```
case 2://edita os seus proprios dados
    printf("Qual o seu id  !?\n-\n");
    scanf("%d", &idprocura);
    limpaBuffer();
    printf("O que deseja atualizar nos seus dados de cliente:\n1-nif\n2-nome\n3-pais\n4-morada\n-");
    scanf("%d", &opu);
    limpaBuffer();
    switch (opu) {
        case 1:
            printf("\nDigite o novo nif\n");
            scanf("%d", &nifnovo);
            limpaBuffer();
            atualizarClienteNif(&loja, &idprocura, nifnovo);
            c = obterCaratere(CARATERE);
            break;
```

Figura 14. Função para edição de dados

```
int atualizarClienteNif(Clientes *clientes, int *idCliente, int nifnovo) {

    int pos;
    procuraCliente(clientes, idCliente, &pos);
    if (pos == -1) return 0;

    clientes->listacliente[pos].nif = nifnovo;

    return 1;
}
```

Figura 15. Exemplo de função de substituição de dados

2.3.2.3 Remoção Pessoal

Obstantes de que pode ser do interesse do cliente um abandono dos serviços da empresa, para que este não ponha em causa os seus dados pessoais, é possível que este se auto remova da lista de clientes (Figura16.).

```
case 3://autoremove se
    printf("Qual o seu id !?\n");
    scanf("%d", &idprocura);
    limpaBuffer();
    printf("Deseja mesmo auto remover-se da lista de clientes 1- se sim ,2-se nao  !?\n- ");
    scanf("%d", &j);
    limpaBuffer();
    if (j == 1) {
        removerCliente(&loja, &idprocura);
    } else {
        printf("Cliente nao removido ");
    }
    c = obterCaratere(CARATERE);
    break;
```

Figura 16. Função para Auto Remoção do Cliente

2.3.2.4 Realizar Encomenda

Por fim, o cliente consegue realizar encomendas. A cada encomenda realizada o id do comprador irá ser armazenado e será associado a um numero de encomenda usados posteriormente para identificar a encomenda e validar que a mesma não é repetida. De notar que o tamanho da encomenda e a sua data de entrega também são armazenados nesta mesma estrutura (Figura17.).

```
typedef struct {
  int numencomenda;
  Produto produtos[10];
  prod *prode;
  Data dataentrega;
  int tamanhoprode;
  int contadorprode;
  int id;
}Encomenda;
```

Figura 17. Struct de dados da encomenda

Para um registo da encomenda, é pedido ao utilizador o seu id, e um número de encomenda á sua escolha sendo depois verificada a sua existência, para que seja possível uma leitura de todos os produtos da empresa, estes são listados para que o cliente possa efetuar a sua compra inserindo apenas o código dos produtos que deseja.

```
    imprimirTodosProdutos(produtos);

    do {
        prod temp;
        printf("Digite o codigo do seu produto  :\n- ");
        scanf("%s", temp.cod);
        limpaBuffer();
        printf("Digite a quantidade do seu produto  :\n- ");
        scanf("%d", &(temp.quantidadeprod));
        limpaBuffer();


        inserirProdutoEncomenda( encomenda,  &temp);
        printf("Deseja adicionar mais algum produto a sua encomenda s-sim n-nao \n-");
        scanf("%c", & d);
        limpaBuffer();

    } while (d != 'n' && d != 'N');
    clientes->listacliente[pos].nencomendas += 1;
}
```

Figura 18.Função de registo de Encomendas

2.3.2.5 Edição de Encomenda

Caso o utilizador cometa algum erro durante o registo da encomenda ou exista uma perda de interesse em algum produto selecionado, o utilizador carece de uma opção de edição de encomenda. O cliente apenas precisa de colocar o seu número de encomenda, esta será a cessada e serão dispostas ao utilizador opções para a edição (Figura 19.).

```
case 4:
    printf("\nQual o numero do produto que deseja editar\n-");
    scanf("%d", &comp);
    limpaBuffer();
    printf("\nDigite a nova quantidade\n");
    scanf("%d", &novaquantidade);
    limpaBuffer();
    editarProdutoQuantidade(&encomendas, &idprocura, comp, novaquantidade);
    c = obterCaratere(CARATERE);
    limpaBuffer();
    break;
case 5:
    printf("\nQual o numero do produto que deseja remover\n-");
    scanf("%d", &comp);
    limpaBuffer();
    removerProd(&encomendas, &idprocura, comp);
    c = obterCaratere(CARATERE);
    limpaBuffer();
    break;
```

Figura 19 Exemplos de opções de edição

2.3.2.6 Remover Encomenda

Por fim é possível ao utilizador cancelar a encomenda, se o mesmo já não se encontrar interessado na totalidade do seu conteúdo. A encomenda será apagada da lista de encomendas através do número de encomenda (Figura 20.).

```
case 3://remover encomenda
    printf("Qual o numero da sua encomenda que quer remover !?\n");
    scanf("%d", &idprocura);
    limpaBuffer();
    printf("Deseja mesmo  remover da lista de a sua encomenda 1- se sim ,2-se nao  !?\n- ");
    scanf("%d", &j);
    limpaBuffer();
    if (j == 1) {
        removerEncomenda(&encomendas, &idprocura);
    } else {
        printf("Encomenda nao removida ");
    }
    c = obterCaratere(CARATERE);
    break;
```

Figura 20. Função para Remover Encomenda.

3.Funcionalidades Propostas

Nesta secção são apresentadas as cinco funcionalidades propostas pelo grupo como consta no documento de exigências á realização do projeto. Estas funcionalidades são de caracter complementar ao programa conferindo á empresa mais capacidade de trabalho, uma melhor gestão por parte do cargo administrativo, segurança na sua utilização e uma melhor experiência de utilização para o cliente.

3.1 Utilizador – "Funcionário"

De modo a otimizar ao máximo o funcionamento da empresa, o grupo propôs a criação de um perfil adicional englobando assim o seu uso aos funcionários.

Estes serão obrigados a recorrer ao programa para registar a sua entrada ou saída do estabelecimento de trabalho (Figura21.), a partir desta operação o programa será capaz de registar numa lista, lista esta á qual o administrador é capaz de aceder.

```
switch (funcionarios) {
    case 0:
        break;
    case 1:
        printf("Diga o que pretende fazer 1-entrada 2-saida!?");
        scanf("%d", &n);
        limpaBuffer();
        if (n == 1) {
            logs(&id, "entrada", LOGS);
            printf("\n Entrada efetuada com sucesso bom trabalho\n");
        } else {
            logs(&id, "saida", LOGS);
            printf("\n Saida efetuada com sucesso tenha uma boa continuação\n");
        }
        c = obterCaratere(CARATERE);
        break;
```

Figura 21 Sistema de registo de entradas e saídas

Este registo é feito através de uma função "logs" (Figura 22.), responsável por escrever no ficheiro a data de entrada ou saída dada pelo funcionário.

```
void logs(int *id,char *msg, char *nomeficheiro) {
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);

    FILE *fp = fopen(nomeficheiro, "a");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fprintf(fp, "%d-%02d-%02d %02d:%02d:%02d - id do funcionario :"
            "%d ,%s\n", tm->tm_year + 1900, tm->tm_mon + 1, tm->tm_mday, tm->tm_hour, tm->tm_min, tm->tm_sec,*id, msg);

    fclose(fp);
}
```

Figura. 22 Função "Logs"

O funcionário consegue também consultar as listagens de produtos, componentes e encomendas.

3.2 Atribuição de Passwords

Tendo em conta que os perfis de Administrador e Funcionário podem consultar dados de clientes ou produtos, os grupos propõem a atribuição de passwords para os cargos com maiores permissões no sistema (Figura 23.).

```
case 1:
    printf("\nDigite a palavra passe de admin\n-");
    scanf("%s", pa);
    limpaBuffer();
    if ((strcmp(pa, passeadmin)) != 0) {
        printf("\nPalavra passe errada\n");
        return 0;
    }
```

Figura 23. Verificação password Admin

```
case 3:
    printf("\nDigite a palavra passe de funcionario\n-");
    scanf("%s", pf);
    if ((strcmp(pf, passefunc)) != 0) {
        printf("\nPalavra passe errada\n");
        return 0;
    }
```

Figura 24. Verificação password Funcionário

Esta verificação é feita sempre que os perfis de Administrador e Funcionário são selecionados. Como é percetível nos exemplos implementados (Figura 23 & 24), esta validação é feita através de uma comparação entre uma *string* inserida pelo utilizador momentâneo e uma *string* declarada previamente (Figura 25.).

```
passeadmin[CAPACIDADE] = "admin123", passefunc[CAPACIDADE] = "func123";
```

Figura 25. Declaração prévia das strings corretas

3.3 Sistema de Presenças

Para um maior controlo por parte do administrador sobre os seus funcionários o grupo propõe a existência de um sistema de presenças e respetivo intervalo de tempo. Posto isto, cada funcionário terá que na hora de entrada marcar a sua entrada onde o programa autonomamente irá atribuir numa lista o horário atual, este mesmo mecanismo é aplicado na sua saída. É possível assim que o administrador quantifique as horas de trabalho de cada trabalhador.

Para esta função *logs* é iniciada uma variável e é lhe atribuído o valor do tempo atual, esta atribuição é feita utilizando funções de bibliotecas pré-definidas (_time_h), de seguida é aberto o ficheiro de registo de presença e são inseridos os dados temporais da entrada e ou saída (Figura 26.).

```
void logs(int *id,char *msg, char *nomeficheiro) {
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);

    FILE *fp = fopen(nomeficheiro, "a");
    if (fp == NULL) {
        exit(EXIT_FAILURE);
    }

    fprintf(fp, "%d-%02d-%02d %02d:%02d:%02d - id do funcionario :"
            "%d ,%s\n", tm->tm_year + 1900, tm->tm_mon + 1, tm->tm_mday, tm->tm_hour, tm->tm_min, tm->tm_sec,*id, msg);

    fclose(fp);
}
```

Figura 26 Função Registo de presença

3.4 Ordenação de Clientes por encomendas

Visando uma leitura mais eficaz dos dados, o grupo propõe em quarto lugar, um sistema de ordenação da lista de clientes. Esta lista apenas é visível ao administrador que por meros aspetos estatísticos pode desejar perceber quem é o seu maior ou menor comprador, para isso é-lhe agora facultada a lista ordenada por ordem decrescente do cliente com mais, ao cliente com menos encomendas. Esta função pode futuramente vigorar na atribuição de promoções aos clientes mais fiéis.

Para a elaboração recorreu-se a uma *bubble sort* onde é percorrida sequencialmente cada par de posições adjacente da lista, estas posições são comparadas e caso não estejam numa ordem correta, é efetuada uma troca entre os mesmos (Figura 27). Este processo é repetido até ser percorrida a lista na sua integra.

```
void trocaClientes(Cliente *cliente1, Cliente *cliente2) {
    Cliente temp = *cliente1;
    *cliente1 = *cliente2;
    *cliente2 = temp;
}

/**
 *
 * @inheritDoc
 */
void OrdenaClientes(Clientes *clientes) {
    int i, j;
    for (i = 0; i < clientes->inseridosc - 1; i++) {
        for (j = 0; j < clientes->inseridosc - i - 1; j++) {
            if (clientes->listacliente[j].nencomendas < clientes->listacliente[j + 1].nencomendas) {
                trocaClientes(&clientes->listacliente[j], &clientes->listacliente[j + 1]);
            }
        }
    }
```

Figura 27. Função para ordenação Clientes/Compras

3.5 Ordenação de Produtos por Preço

De uma natureza similar á enunciada anteriormente, é proposta uma ordenação dos produtos através do seu preço, esta ordenação permite ao cliente uma visualização dos produtos ordenada por ordem decrescente do maior ao menor preço (Figura 28.). O método utilizado para a ordenação é também semelhante ao implementado na ordenação dos clientes usando encomendas.

```
void trocaProdutos(Produto *produto1, Produto *produto2) {
    Produto temp = *produto1;
    *produto1 = *produto2;
    *produto2 = temp;
}

/**
 *
 * @inheritDoc
 */
void OrdenaProdutos(Produtos *produtos) {
    int i, j;
    for (i = 0; i < produtos->contadorp - 1; i++) {
        for (j = 0; j < produtos->contadorp - i - 1; j++) {
            if (produtos->listaProdutos[j].preco < produtos->listaProdutos[j+1].preco) {
                trocaProdutos(&produtos->listaProdutos[j], &produtos->listaProdutos[j + 1]);
            }
        }
    }

    imprimirTodosProdutos(produtos);
}
```

Figura 28 Função para ordenação Produtos/Preço

4.Estrutura analítica do projeto

O planeamento do projeto foi concebido de forma precoce com especial atenção ao tempo necessário á sua elaboração em relação á data de entrega, neste planeamento esteve também em conta o ritmo de instrução por parte das unidades curriculares competentes, visto que graças a motivos de calendário as aulas foram atrasadas.

Com este atraso o grupo teve uma necessidade de adaptação do planeamento incidindo-se inicialmente em funções de simples implementação, como a gestão de clientes e produtos sem memória dinâmica ou persistência de dados associada.

Durante a implementação do primeiro perfil do programa, no caso "Administrador", o grupo optou pela seguinte ordem de conceção:

Administrador:

1. Criação de uma biblioteca de estruturas essenciais ao armazenamento e manipulação de dados.
2. Implementação da estrutura dos menus em função do esquema de funções CRUD.

Após esta implementação o grupo depara-se com o facto de que os outros perfis idealizados (Cliente & Funcionário) apresentam um caracter semelhante ao implementado no Administrador. Por isso foi também criada uma estruturação para os restantes perfis, permitindo assim um planeamento mais simples.

Com a estruturação dos menus passou-se a idealizar as restantes estruturas de dados necessárias às restantes funções.

Com a sucessiva testagem do programa originou-se uma necessidade de criação de listagens para as diferentes funções de modo a que seja possível a verificação do output dado por uma função na lista.

Com o decorrer das atividades letivas o grupo começou a ser introduzido a noções de memória dinâmica e utilização de ficheiros. Levando a que a implementação destas funções fosse o próximo passo a seguir pelo grupo.

Dadas as dificuldades na sua implementação estas funcionalidades precisaram de bem mais do que o tempo estipulado para a sua aplicação, e visto que a data de entrega se aproximava, o grupo decidiu repartir-se para a aplicação de estruturas de verificação e a continua aplicação das funcionalidades implementadas.

Após o esclarecimento junto do docente da unidade curricular, a aplicação de memória dinâmica foi bem sucedida, persistindo no entanto dificuldades na aplicação de ficheiros.

De modo a maximizar a produtividade do grupo, mesmo sem todas as funções operacionais o grupo incidiu-se nas seguintes tarefas e objetivos:

1. Implementação das funcionalidades propostas pelo grupo.

2. Formatação e documentação do código.

3. Finalização de Relatório do Projeto.

Mesmo com as considerações finais escritas, o grupo insistiu na implementação das funções em falta até ao final do prazo, sendo, no entanto, o seu resultado pouco positivo.

O grupo apresentou durante o desenvolvimento do projeto uma estrutura positivamente cooperativa, com uma investigação ativa sobre as matérias relativas e necessárias ao projeto. Foi praticado um trabalho igualitário entre os membros do grupo em perspetiva o desenvolvimento das suas competências pessoais.

5. Funcionalidades Implementadas

Além das funções consideradas base á elaboração do projeto, são necessárias também funções de maior complexidade que implicam conhecimentos ditos avançados.

5.1 Memória Dinâmica

Para tornar o projeto mais otimizado foi proposta a implementação de memória dinâmica, funcionalidade este que permite ao programa implementado, uma otimização de armazenamento.

A memória dinâmica permite ao programa uma automática disponibilização e eliminação de memória pronta ao seu funcionamento.

Estas funcionalidades foram implementadas nas componentes Crud, onde antes de cada operação existe uma verificação de que a memória necessária á operação existe, caso contrário a memória disponível irá sofrer uma alteração ou estes dados são realocados na memória.

5.2 Persistência de Dados

Outra das funcionalidades implementadas foi a manipulação de ficheiros, permitindo assim que o programa seja capaz de uma leitura e escrita de dados com suporte de ficheiros anexos.

A manipulação de ficheiros permite ao programa uma persistência de dados, possibilitando registos e impressão de resultados por parte dos algoritmos do programa. Exemplos práticos são a impressão de listas de dados por parte do utilizador, no caso é possível o display de listas de componentes ou produtos.

6. Conclusão

Este capítulo termina o relatório, apresentando um balanço geral do projeto desenvolvido. Desta forma, inicia-se pela discussão dos objetivos alcançados, bem como as limitações e trabalho a desenvolver futuramente que a solução suscitou. Por fim, é fornecida uma opinião critica, de caracter coletivo, acerca do trabalho desenvolvido.

6.1 Objetivos Concretizados

Os objetivos do projeto foram, de forma geral, atingidos. Trata-se de um trabalho de desenvolvimento de capacidades na área da programação de diversos fundamentos, o que no decorrer do seu desenvolvimento levou á recalendarização de prazos e definição de diferentes metas.

A implementação de uma estrutura base de CRUD foi capaz de atender aos requisitos propostos. No entanto, o desenvolvimento por vezes tornou-se complexo visto a necessária implementação de funcionalidades como memória dinâmica e leitura/escrita de ficheiros, funcionalidades estas que merecem por parte do programador um conhecimento mais alargado.

O planeamento e desenvolvimento das funções de leitura e escrita em ficheiros, revelaram-se, para a persistência de dados, essenciais.

Como evidenciado na secção seguinte (secção 6.2), a implementação de persistência de dados não foi bem sucedida, sendo a sua aplicação parcial. Por conseguinte o grupo recorreu a formas de testagem diferentes com o objetivo de fornecer ao utilizador uma experiência de uso de aproximada qualidade.

Assim, conclui-se que o trabalho desenvolvido foi no entanto positivo.

6.2 Limitações e Condicionantes

Apesar de todo o esforço na concretização de todos os objetivos propostos pelo trabalho, os imprevistos surgem e comprometem o desenvolvimento. Grande parte dos requisitos foram concluídos com sucesso, embora, devido a influentes razões ao nível do aluno, ocorreu uma demora imprevista na elaboração do projeto.

O trabalho desenvolvido apresenta um nível de conceitos e processos com investigação passível ao nível do aluno, investigação esta necessária, graças a problemas na instrução de conteúdos na unidade curricular responsável pela fundamentação teórica "Fundamentos da Programação".

Dadas estas dificuldades, como grupo desejamos agradecer ao docente responsável pelas aulas de "Laboratórios de Programação", pela disponibilidade para esclarecimento de dúvidas e síntese de conteúdos para os alunos desfavorecidos pela situação previamente descrita. É no entanto lamentável perceber que nem todos os objetivos foram cumpridos contrariamente á vontade do grupo na sua resolução.

Conclui-se assim que, para além de todo o trabalho desenvolvido com sucesso, existiram condicionantes que implicaram uma maior investigação autónoma.

6.3 Apreciação Final

Esta secção visa expor a apreciação coletiva e critica, experienciada ao longo da elaboração do trabalho proposto.

A unidade curricular LP é determinante na afirmação de qualquer estudante de LEI da ESTG. Esta possibilita a aplicação de competências técnicas e soft skills necessárias ao percurso académico, elevando-as, levando a aquisição de competências como cooperação, gestão e técnicas de programação.

O projeto selecionado pela UC, estava inserido na área dos sistemas de gestão empresarial, revelando-se um caracter enriquecedor a prováveis aplicações futuras. Esta é uma área onde, regularmente, surgem complexos desafios da engenharia informática.

Em conclusão, o grupo usufrui agradavelmente do projeto proposto pela unidade curricular, onde expandiu e diversificou os conhecimentos na temática escolhida.