-Relatório Técnico do Projeto da Unidade Curricular de Fundamentos de Desenvolvimento de Software

Luís Martins 1221432, Daniel Nogueira 1221434, Fernando Procópio 1210874

Grupo 3
Turma: 1DC

Resumo: Este relatório tem como propósito fornecer um suporte escrito da análise e desenvolvimento da parte inicial do desenvolvimento da aplicação que utiliza a linguagem de programação C++. O objetivo deste relatório é esclarecer os trechos de código mais importantes e explicar algumas das decisões tomadas. A elaboração deste tipo de relatório é importante para garantir que todos possam compreender o software e sua implementação.

1 Introdução

O foco de nosso projeto é o desenvolvimento de uma aplicação destinada à gestão de uma loja de informática, que abarca clientes, funcionários e gerentes. Cada identidade desempenha funções específicas dentro do contexto da loja, as quais serão previamente definidas por nós. A nossa aplicação proporcionará aos utilizadores a capacidade de criar uma conta e, a partir daí, ver, comparar preços e fazer encomendas. No caso dos funcionários, eles vão ter permissão para editar o stock e os preços dos produtos, além de gerir as encomendas. A responsabilidade pela gestão dos funcionários será atribuída aos gerentes, que vão poder adicionar e remover membros da equipa.

2 Implementação

O método `runLogin` gerencia o processo de login, capturando o nome de usuário e a senha do cliente, verificando a existência do usuário na lista de clientes e validando a senha. Se o login for bem-sucedido, o cliente é autenticado e o controle é passado para o próximo passo do fluxo de trabalho. Caso contrário, mensagens de erro apropriadas são exibidas.

```
void Controller::runLogin() {
    //Username input
    string username = this->clientView.getUsername();
    ClientList &clientList = this->model.getClientList();
    Client *tempClient = clientList.getByUsername( &: username);

    //Checking if user exists
    if (tempClient == nullptr) {
        this->clientView.invalidUsername();
        return;
    }
    string password = this->clientView.getPassword();

    //Checking if password matches
    if (tempClient->doesPasswordMatch( &: password)) {
        this->login( pClient: (Client *) tempClient);
        cout << "Login Successful";
        runClient();
    } else {
        this->clientView.incorrectPassword();
    }
}
```

Figure 1 – Função para verificar Login

O método runCreateAccount gerencia o processo de criação de uma nova conta de usuário, recuperando a lista de clientes existentes, captura o nome de usuário desejado e verifica se já está em uso. Se já estiver em uso, exibe uma mensagem de erro e interrompe o processo. Ao inserir um nome de usuário valido, o programa captura a senha do usuário e adiciona o novo cliente a lista de clientes.

```
void Controller::runCreateAccount() {
    ClientList &clientList = this->model.getClientList();
    string username = this->clientView.getUsername();
    if (clientList.getByUsername( &: username) != nullptr) {
        cout << "Username already taken.";
        return;
    }
    int permission = 0;
    string password = this->clientView.getPassword();
    Client newClient( &: username,  &: password,  permission: 0);
    clientList.addClient( &: newClient);
}
```

Figure 2 – Código para criar contas

O método `runCart` gerencia as operações relacionadas ao carrinho de compras, como verificar o valor total, imprimir os itens do carrinho, deletar itens do carrinho e fazer o checkout. Obtém a lista de produtos do carrinho, verifica se o carrinho está vazio; se estiver, notifica o usuário e encerra o método. Entra em loop permitindo ao usuário interagir com o carrinho, oferecendo opções para deletar produtos ou fazer checkout, atualizando e imprimindo o carrinho após cada operação. Se o carrinho ficar vazio após uma operação ou escolhe a opção de sair, sai do loop e encerra o método.

```
void Controller::runCart() {
    list<Product> cartProducts = this->cart.getProducts();

    //Checking if cart has any product
    if (this->cart.getValue() == 0) {
        this->cartView.empty();
        return;
    }

    cout << "Cart value: " << this->cart.getValue() << endl;
    this->cartView.printCartClient( &: cartProducts);
    int op = -1;
    do {
        op = this->view.menuCart();
        switch (op) {
            case 1: runDeleteFromCart(); // 1 - Delete from cart
                if (this->cart.getValue() == 0) {
                    op = 0;
                    break;
                }
                cartProducts = this->cart.getProducts();
                this->cartView.printCartClient( &: cartProducts);
                break;
            case 2: // 2 - Checkout
                if (this->cart.getValue() == 0) {
                    this->cartView.empty();
                    break;
                }
                checkOutClient();
                if (this->cart.getValue() == 0) {
                    op = 0;
                    break;
                }
                break;
            default:
                break;
        }
    } while (op != 0);
}
```

Figure 3 – Código de funcionamento do carrinho

O método `runProducts` gerencia as operações relacionadas à visualização e manipulação de produtos disponíveis, como adicionar produtos ao carrinho e visualizar o carrinho, obtendo a lista de produtos disponíveis e exibindo-a. O método entra em loop permitindo ao usuário adicionar produtos e visualizar o carrinho, imprimindo e atualizando a cada operação e sai do loop, assim encerrando o método, quando o usuário escolhe a opção de sair ou visualizar o carrinho.

```
void Controller::runProducts() {
    int op = -1;
    ProductList *modelProductListPointer = &this->model.getProductList();
    ProductList availableProductList = modelProductListPointer->getAvailable();
    list<Product> clientProducts = availableProductList.getAll();
    list<Product> allProducts = modelProductListPointer->getAll();

    this->productView.printProductListClient( &: clientProducts,  listTitle: "Products List:");


    do {
        op = this->view.menuProducts();
        switch (op) {
            case 1:
                this->productView.printProductListClient( &: clientProducts,  listTitle: "Products List: "); //1 - Add to carts
                runAddToCart();
                break;
            case 2: //2 - View cart
                if (this->cart.getValue() == 0) {
                    this->cartView.empty();
                    op = -1;
                    break;
                }
                runCart();

                op = 0;
                break;
            default:
                break;
        }
    } while (op != 0);
}
```

Figure 4 – Código para manipular e visualizar produtos

3 Testes

Nosso grupo se deparou com dificuldades e erros e não fomos capazes de fazer a implementação do Google Testing Framework no nosso projeto.

3.1 Taxa de sucesso dos testes

Test	Percentage	Notes
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------
	100	------