Sistema de Acesso para Restaurante Universitário

Ana Laura Soares Mello,
Larissa Roncali Faria,
João Pedro Braga da Silva,
Matheus Magalhães Caldeira Brant.

01

Contexto

Introdução

Estamos aqui para apresentar um projeto que visa melhorar a experiência dos estudantes no Restaurante Universitário da UFMG. Com o crescimento das filas e o estresse associado, nossa proposta é criar um sistema que torna o pagamento mais rápido e eficiente. Todos nós já passamos por isso: filas enormes nos Restaurantes Universitários, especialmente na hora do almoço. Esse cenário pode ser frustrante e desnecessário, pois impacta diretamente o dia a dia dos estudantes. Nosso projeto tem como objetivo desenvolver um sistema que automatiza e organiza o processo de pagamento das refeições, contribuindo para a eficiência e satisfação dos estudantes.

02

Sistema Proposto

Funcionalidades principais

O sistema proposto permitirá pagamentos mais rápidos, evitando a necessidade de fila para cartões de débito, e melhorando a fluidez no restaurante. Com o novo sistema, os usuários terão uma experiência muito mais tranquila ao se alimentarem. A agilidade nas transações vai permitir que os estudantes tenham tempo para outras atividades do mundo acadêmico ou de lazer.

03

Código em C

Classe Cliente

```
/**
 * @brief Registra um acesso (refeição), atualizando a data correspondente.
 * @param tipoDeRefeicao Um char indicando o tipo de refeição ('a' para almoço, 'j' para jantar).
 * @return Retorna um booleano indicando se a operação foi bem-sucedida.
 */
bool registrarAcesso(char tipoDeRefeicao);
```

Cliente.hpp

Classe Cliente

```cpp
bool Cliente::registrarAcesso(char tipoDeRefeicao) {
    Data hoje; // Pega a data atual

    // Verifica se o cartão está bloqueado
    if (this->getIsBloqueado()) {
        std::cout << "ERRO: Acesso negado. O cartão está bloqueado." << std::endl;
        return false;
    }

    // Verifica se o cliente tem saldo suficiente
    if (this->getSaldo() < this->getValorRefeicao()) {
        std::cout << "ERRO: Acesso negado. Saldo insuficiente." << std::endl;
        return false;
    }
```

Cliente.cpp

```cpp
    // Lógica para registrar almoço ou jantar
    switch (tipoDeRefeicao) {
        case 'a':
            if (this->getUltimoAlmoco() == hoje) {
                std::cout << "ERRO: Acesso negado. Almoço já registrado hoje." << std::endl;
                return false;
            }
            this->setUltimoAlmoco(hoje);
            break;
        case 'j':
            if (this->getUltimoJantar() == hoje) {
                std::cout << "ERRO: Acesso negado. Jantar já registrado hoje." << std::endl;
                return false;
            }
            this->setUltimoJantar(hoje);
            break;
        default:
            std::cout << "ERRO: Tipo de refeição não reconhecido." << std::endl;
            return false;
    }

    // Deduz o valor da refeição do saldo
    this->setSaldo(this->getSaldo() - this->getValorRefeicao());
    return true; // Retorna true se tudo deu certo
}
```

Cliente.cpp

Classe Funcionário

```
/**
 * @brief Processa a liberação de uma refeição para um cliente.
 * @param cliente Um ponteiro para o objeto Cliente que receberá a refeição.
 * @note Este método contém a lógica de verificação de saldo, bloqueio e registro de acesso.
 */
void liberarRefeicao(Cliente* cliente);
```

Funcionario.hpp

Classe Funcionário

```
/**
 * @brief Processa a liberação de uma refeição para um cliente.
 * @param cliente Um ponteiro para o objeto Cliente que receberá a refeição.
 * @note Este método contém a lógica de verificação de saldo, bloqueio e registro de acesso.
 */
void liberarRefeicao(Cliente* cliente);
```

Funcionario.hpp

Classe Funcionário

```cpp
void Funcionario::liberarRefeicao(Cliente* cliente) {
    try {
        if (cliente->getSaldo() < cliente->getValorRefeicao()) {
            throw ExcecaoSaldoInsuficiente();
        }

        // A lógica de verificação de data esta encapsulada dentro de cliente->registrarAcesso().
        bool sucesso = cliente->registrarAcesso(GerenciamentoDeSistema::_tipoDeRefeicao);

        if (sucesso) {
            std::cout << "Refeição liberada com sucesso pelo funcionário." << std::endl;
        }

    } catch (const ExcecaoSaldoInsuficiente& e) {
        std::cerr << e.what() << std::endl;
    }
}
```

Funcionario.cpp

Classe Aluno

```
/**
 * @brief Define o nível FUMP do aluno e atualiza o valor da refeição de acordo.
 * * O valor da refeição é ajustado com base em uma tabela interna de preços
 * correspondente a cada nível.
 * @param nivel O novo nível FUMP a ser atribuído.
 */
void setNivelFump(int nivel);
```

Aluno.hpp

```cpp
void Aluno::setNivelFump(int nivel) {
    // A estrutura switch é utilizada para mapear o nível FUMP
    // a um valor de refeição específico.
    switch (nivel) {
    case 0:
        this->setValorRefeicao(5.60);
        break;
    case 1:
        this->setValorRefeicao(0.0);
        break;
    case 2:
    case 3: // Níveis 2 e 3 compartilham o mesmo valor
        this->setValorRefeicao(1.0);
        break;
    case 4:
        this->setValorRefeicao(2.0);
        break;
    case 5:
        this->setValorRefeicao(2.90);
        break;
    default:
        // Se o nível for inválido, uma mensagem é exibida e o nível atual não é alterado.
        std::cout << "Nível FUMP inválido. O valor da refeição não foi alterado." << std::endl;
        return;
    }
    // O nível só é atualizado se for um valor válido.
    _nivelFump = nivel;
}
```

Aluno.cpp

Main

```
do {
  cout << "\n========================================" << endl;
  cout << "        BEM-VINDO AO RU UNIVERSITÁRIO   " << endl;
  cout << "========================================" << endl;
  cout << "Escolha uma opção:" << endl;
  cout << " [1] Cadastrar" << endl;
  cout << " [2] Entrar no Restaurante" << endl;
  cout << " [3] Adicionar Credito" << endl;
  cout << " [4] Verificar Informação" << endl;
  cout << " [5] Salvar Dados" << endl;
  cout << " [6] Sair" << endl;
  cout << "----------------------------------------" << endl;
  cout << "Opção: ";
  cin >> opcao;
```

Main

```
switch (opcao) {
    case 1:
        //int opcao_cadastro;
        cout << "\n>> Selecione o tipo de usuário para cadastro:\n";
        cout << " [1] Aluno\n";
        cout << " [2] Professor\n";
        cout << " [3] Visitante\n";
        cout << " [4] Funcionário\n";
        cout << " [5] Caixa\n";
        cout << " [6] AdmFump\n";
        cout << "---------------------------------------\n";
        cout << "Tipo: ";
        cin >> opcao_cadastro;
        system("cls");                                          // LIMPA A TELA
```

Main

```
if (opcao_cadastro == 1) {
    //string nome_Aluno, cpf_Aluno, curso_Aluno;
    //int nivelFump_Aluno;

    cout << "\n=== Cadastro de Aluno ===" << endl;
    cout << "Nome: ";
    cin.ignore(); // limpa buffer antes do getline
    getline(cin, nome_Aluno);
    cout << "CPF: ";
    getline(cin, cpf_Aluno);
    cout << "Nível FUMP (ex: 1, 2, 3): ";
    cin >> nivelFump_Aluno;
    cin.ignore(); // limpa antes de novo getline
    cout << "Curso: ";
    getline(cin, curso_Aluno);

    Cliente* cliente_Aluno = new Aluno(nome_Aluno, cpf_Aluno,nivelFump_Aluno, curso_Aluno);

    sistema.cadastrarCliente(cliente_Aluno);                    // ATENÇÃO
    cout << "\n>> Aluno cadastrado com sucesso!\n";
} else {
    cout << "\n>> Tipo de cadastro ainda não implementado.\n";
}
system("cls");                                                  // LIMPA A TELA
```

04

Resultados Esperados

Redução de filas

Esperamos uma significativa redução nas filas nos Restaurantes Universitários, uma vez que o tempo de espera para pagamentos será drasticamente reduzido.

Conclusões

Em síntese, nosso sistema tem o potencial de transformar a experiência alimentar dos estudantes da UFMG, tornando-a mais rápida, prática e agradável.