PONTIFÍCIA UNIVERSIDADE CATÓLICA DE

MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA

UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE

Bacharelado em Engenharia de Software

Nome dos integrantes do grupo:

• André Luiz Segato

• João Pedro Fernandes

Nome do sistema:

• Sistema de Controle do Hotel Descanso Garantido

Apresentação:

O sistema de controle do Hotel Descanso Garantido tem como objetivo gerenciar as reservas, clientes e funcionários do hotel, eliminando a necessidade de planilhas e cadernos. O sistema previne a dupla reserva de quartos, organiza a gestão de estadias e facilita a administração dos dados de clientes e funcionários.

Evolução do Backlog do Produto

Inicialmente, organizamos o backlog do produto com as funções básicas do sistema. Cada função foi de responsabilidade de um membro do grupo e foi desenvolvida em sprints de 2 a 3 dias. Seguem as atividades realizadas nas sprints:

Sprint 1

1. Declarar assinatura das funções.

• Responsável: André Luiz

• Descrição: Definição dos parâmetros de entrada e saída das funções principais.

• Funções: cadastrarCliente, cadastrarFuncionario, cadastrarEstadia, pesquisarCliente, pesquisarFuncionario,

mostrarEstadiasCliente, pesquisarQuarto.

2. Documentar as funções indicando sua finalidade, parâmetros de entrada e saída.

• Responsável: João Pedro

• Descrição: Elaborar documentação detalhada das funções.

• Funções: cadastrarCliente, cadastrarFuncionario,

cadastrarEstadia, pesquisarCliente, pesquisarFuncionario,

mostrarEstadiasCliente, pesquisarQuarto.

Sprint 2

3. Implementar o caso de sucesso das funções.

• Responsável: André Luiz

• Descrição: Implementação inicial das funções focando no caso de

sucesso.

o Funções: cadastrarCliente, cadastrarFuncionario,

cadastrarEstadia.

4. Selecão dos casos de teste para assegurar o funcionamento das funções.

o Responsável: João Pedro

o Descrição: Selecionar casos de teste afim de verificar o funcionamento

correto das funções.

o Funções: cadastrarCliente, cadastrarFuncionario,

cadastrarEstadia.

Sprint 3

5. Execução dos casos de teste planejados anteriormente para as funções.

o Responsável: André luiz

o Descrição: Execução manual e automatizada dos casos de teste usando a

biblioteca munit.

o Funções: cadastrarCliente, cadastrarFuncionario,

cadastrarEstadia.

6. Elaborar o relatório da execução de testes.

o Responsável: João Pedro

o Descrição: Criação de um relatório detalhado contendo os casos de teste,

saídas esperadas e resultados reais.

o Funções: cadastrarCliente, cadastrarFuncionario,

cadastrarEstadia.

7. Implementar os casos especiais e exceções.

o Responsável: André luiz

o Descrição: Implementação de tratamento de exceções e casos especiais

nas funções.

o Funções: cadastrarCliente, cadastrarFuncionario,

cadastrarEstadia.

Documentação das Funcionalidades do Software

Bibliotecas Utilizadas

• stdio.h: Funções de entrada e saída.

• stdlib.h: Funções utilitárias de propósito geral, como alocação de memória.

• string.h: Funções para manipulação de strings.

• time.h: Funções para manipulação de datas e tempos.

• munit.h: Biblioteca para execução de testes unitários.

Descrição das Funções:

• cadastrar_cliente

Propósito: Cadastra um novo cliente no sistema.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• cadastrar_estadia

Propósito: Cadastra uma nova estadia no sistema.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• cadastrar_funcionario

Propósito: Cadastra um novo funcionário no sistema.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• listar_clientes

Propósito: Lista todos os clientes cadastrados.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• listar_estadias

Propósito: Lista todas as estadias cadastradas.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• listar_funcionarios

Propósito: Lista todos os funcionários cadastrados.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

Funções de Utilidade

• verificar_disponibilidade_quarto

Propósito: Verifica se há disponibilidade de um quarto para a quantidade de hóspedes especificada.

Parâmetros:

int quantidade_hospedes: Quantidade de hóspedes.

int *numero_quarto: Ponteiro para armazenar o número do quarto disponível.

Valor de Retorno: int: Retorna 1 se houver um quarto disponível, caso contrário retorna 0.

•dataValida

Propósito: Verifica se uma data é válida.

Parâmetros:

int dia: Dia da data.

int mes: Mês da data.

int ano: Ano da data.

Valor de Retorno: int: Retorna 1 se a data for válida, caso contrário retorna 0.

• calcular_dias

Propósito: Calcula o número de dias entre duas datas.

Parâmetros:

Data entrada: Data de entrada.

Data saida: Data de saída.

Valor de Retorno: int: Número de dias entre as datas.

calcular_valor_estadia

Propósito: Calcula o valor total de uma estadia com base na data de entrada, data de saída e o valor diário da estadia.

• Parâmetros:

Data entrada: Estrutura contendo a data de entrada.

Data saida: Estrutura contendo a data de saída.

float valor_diaria: Valor da diária.

Valor de Retorno: float: Valor total da estadia.

• gerar_codigo_cliente

Propósito: Gera um novo código para um cliente.

Parâmetros: Nenhum.

Valor de Retorno: int: Código gerado para o cliente.

• gerar_codigo_funcionario

Propósito: Gera um novo código para um funcionário.

Parâmetros: Nenhum.

Valor de Retorno: int: Código gerado para o funcionário.

• salvar_funcionarios_txt

Propósito: Salva os dados dos funcionários em um arquivo de texto.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• carregar_funcionarios

Propósito: Carrega os dados dos funcionários de um arquivo de texto.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• salvar_clientes_txt

Propósito: Salva os dados dos clientes em um arquivo de texto.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• carregar_clientes

Propósito: Carrega os dados dos clientes de um arquivo de texto.

Parâmetros: Nenhum.

Valor de Retorno: Nenhum.

• mostrarMensagemSeVazio

Propósito: Mostra uma mensagem se a quantidade de um determinado item for zero.

Parâmetros:

int quantidade: Quantidade de itens.

• const char *mensagem: Mensagem a ser exibida.

Valor de Retorno: Nenhum.

Estruturas de Dados

• Além das funções, o código também define várias estruturas de dados e variáveis globais:

Estruturas:

• Cliente: Representa um cliente com campos como código, nome, endereço, telefone e código do funcionário responsável.

• Funcionario: Representa um funcionário com campos como código, nome, cargo e salário.

• Data: Representa uma data com campos de dia, mês e ano.

• Estadia: Representa uma estadia com campos como datas de entrada e saída, código do cliente, código do funcionário, quantidade de hóspedes, número do quarto e valor total.

Variáveis Globais:

•Cliente clientes[MAX_CLIENTES]: Array de clientes.

• int countClientes: Contador de clientes.

• Funcionario funcionarios[MAX_FUNCIONARIOS]: Array de funcionários.

• int countFuncionarios: Contador de funcionários.

• Estadia estadias[MAX_ESTADIAS]: Array de estadias.

• int countEstadias: Contador de estadias.

Constantes:

•MAX_CLIENTES, MAX_FUNCIONARIOS, MAX_ESTADIAS, MAX_QUARTOS, VALOR_DIARIA: Definem limites e valores padrão para o sistema.

Planejamento dos Casos de Teste

Entradas:

• Dados válidos e inválidos para cliente, funcionário e estadia.

• Datas de entrada e saída.

Procedimento de Teste:

• Inserir dados válidos e verificar se o sistema aceita.

• Inserir dados inválidos e verificar se o sistema rejeita e solicita nova entrada.

• Verificar a contagem de dias e cálculo do valor da estadia.

Saídas Esperadas:

• Sucesso na inserção de dados válidos.

• Mensagens de erro e solicitação de nova entrada para dados inválidos.

• Correta contagem de dias e cálculo do valor total da estadia.

Implementação dos Casos de Teste Automatizados

Planejamento dos Casos de Teste

Entrada:

• Dados válidos e inválidos para cliente, funcionário e estadia.

• Datas de entrada e saída.

```
#include "munit.h"
```

• // Teste para validar a função calcular_valor_estadia

```
float calcular_valor_estadia(Data entrada, Data saida, float valor_diaria) {
    if (!dataValida(entrada.dia, entrada.mes, entrada.ano) || !dataValida(saida.dia, saida.mes, saida.ano)) {
        return -1;
    }
    int dias = calcular_dias(entrada, saida);
    return dias * valor_diaria;
}
```

• // Teste para validar a função verificar_disponibilidade_quarto

```
#include "munit.h"
int verificar_disponibilidade_quarto(int quantidade_hospedes, int *numero_quarto);
static MunitResult
test_verificar_disponibilidade_quarto(const MunitParameter params[], void* user_data) {
    int numero_quarto;
```

```
    munit_assert_int(verificar_disponibilidade_quarto(2, &numero_quarto), ==, 1);
    munit_assert_int(numero_quarto, >=, 1);
    munit_assert_int(numero_quarto, <=, 50);
    munit_assert_int(verificar_disponibilidade_quarto(5, &numero_quarto), ==, 1);
    munit_assert_int(numero_quarto, >=, 1);
    munit_assert_int(numero_quarto, <=, 50);
    munit_assert_int(verificar_disponibilidade_quarto(0, &numero_quarto), ==, 0);

    return MUNIT_OK;
}
```

• //Teste para verificar a função dataValida:

```
#include "munit.h"

int dataValida(int dia, int mes, int ano);

static MunitResult
test_dataValida(const MunitParameter params[], void* user_data) {
    munit_assert_int(dataValida(29, 2, 2024), ==, 1);

    munit_assert_int(dataValida(30, 2, 2023), ==, 0);

    munit_assert_int(dataValida(31, 4, 2024), ==, 0);

    return MUNIT_OK;
}
```

// Testes automatizados para cada uma das funções do código:

• Teste para calcular_valor_estadia

```
static MunitResult
test_calcular_valor_estadia(const MunitParameter params[], void* user_data) {
    Data entrada1 = {1, 1, 2024};
    Data saida1 = {5, 1, 2024};
    munit_assert_float(calcular_valor_estadia(entrada1, saida1, 500.0), ==, 2000.0);

    Data entrada2 = {15, 7, 2024};
    Data saida2 = {20, 7, 2024};
    munit_assert_float(calcular_valor_estadia(entrada2, saida2, 500.0), ==, 2500.0);

    Data entrada3 = {28, 12, 2023};
    Data saida3 = {2, 1, 2024};
    munit_assert_float(calcular_valor_estadia(entrada3, saida3, 500.0), ==, 2500.0);

    Data entrada4 = {31, 2, 2024};  // Data inválida
    Data saida4 = {5, 3, 2024};
    munit_assert_float(calcular_valor_estadia(entrada4, saida4, 500.0), ==, -1.0);

    return MUNIT_OK;
}
```

• Teste para verificar_disponibilidade_quarto

```
static MunitResult
test_verificar_disponibilidade_quarto(const MunitParameter params[], void* user_data) {
    int numero_quarto;

    munit_assert_int(verificar_disponibilidade_quarto(2, &numero_quarto), ==, 1);
    munit_assert_int(numero_quarto, >=, 1);
    munit_assert_int(numero_quarto, <=, 50);

    munit_assert_int(verificar_disponibilidade_quarto(5, &numero_quarto), ==, 1);
    munit_assert_int(numero_quarto, >=, 1);
    munit_assert_int(numero_quarto, <=, 50);

    munit_assert_int(verificar_disponibilidade_quarto(0, &numero_quarto), ==, 0);

    return MUNIT_OK;
}
```

• Teste para dataValida

```
static MunitResult
test_dataValida(const MunitParameter params[], void* user_data) {
    munit_assert_int(dataValida(29, 2, 2024), ==, 1);
    munit_assert_int(dataValida(30, 2, 2023), ==, 0);
    munit_assert_int(dataValida(31, 4, 2024), ==, 0);
    munit_assert_int(dataValida(1, 1, 2023), ==, 1);

    return MUNIT_OK;
}
```

• Teste para calcular_dias

```
static MunitResult
test_calcular_dias(const MunitParameter params[], void* user_data) {
    Data entrada1 = {1, 1, 2024};
    Data saida1 = {5, 1, 2024};
    munit_assert_int(calcular_dias(entrada1, saida1), ==, 4);

    Data entrada2 = {15, 7, 2024};
    Data saida2 = {20, 7, 2024};
    munit_assert_int(calcular_dias(entrada2, saida2), ==, 5);

    Data entrada3 = {28, 12, 2023};
    Data saida3 = {2, 1, 2024};
    munit_assert_int(calcular_dias(entrada3, saida3), ==, 5);

    return MUNIT_OK;
}
```

• Teste para gerar_codigo_cliente e gerar_codigo_funcionario

```c
static MunitResult
test_gerar_codigo_cliente(const MunitParameter params[], void* user_data) {
    int codigo1 = gerar_codigo_cliente();
    int codigo2 = gerar_codigo_cliente();
    munit_assert_int(codigo1, !=, codigo2);

    return MUNIT_OK;
}

static MunitResult
test_gerar_codigo_funcionario(const MunitParameter params[], void* user_data) {
    int codigo1 = gerar_codigo_funcionario();
    int codigo2 = gerar_codigo_funcionario();
    munit_assert_int(codigo1, !=, codigo2);

    return MUNIT_OK;
}
```

Relatório de Execução de Testes

Função	Caso de Teste	Entrada	Saída Esperada	Saída Real	Resultado
`calcular_valor_estadia`	1	{1, 1, 2024}, {5, 1, 2024}, 500.0	2000.0	2000.0	Passou
`calcular_valor_estadia`	2	{15, 7, 2024}, {20, 7, 2024}, 500.0	2500.0	2500.0	Passou
`calcular_valor_estadia`	3	{28, 12, 2023}, {2, 1, 2024}, 500.0	2500.0	2500.0	Passou
`calcular_valor_estadia`	4	{31, 2, 2024}, {5, 3, 2024}, 500.0	-1	-1	Passou
`verificar_disponibilidade_quarto`	1	2, &numero_quarto	1	1, 1 <= numero_quarto <= 50	Passou
`verificar_disponibilidade_quarto`	2	5, &numero_quarto	1	1, 1 <= numero_quarto <= 50	Passou
`verificar_disponibilidade_quarto`	3	0, &numero_quarto	0	0	Passou
`dataValida`	1	29, 2, 2024	1	1	Passou
`dataValida`	2	30, 2, 2023	0	0	Passou
`dataValida`	3	31, 4, 2024	0	0	Passou
`calcular_dias`	1	{1, 1, 2024}, {5, 1, 2024}	4	4	Passou
`calcular_dias`	2	{15, 7, 2024}, {20, 7, 2024}	5	5	Passou
`calcular_dias`	3	{28, 12, 2023}, {2, 1, 2024}	5	5	Passou
`gerar_codigo_cliente`	1		Código único	Código único	Passou
`gerar_codigo_funcionario`	1		Código único	Código único	Passou

Código Completo do Programa

/*

==========
=

Nome : Hotel Descanso Garantido

Autores :João Pedro Fernandes e André Luiz Segato

Trabalho : Projeto AEDS e FES

==========
=

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

 #define MAX_CLIENTES 100
 #define MAX_FUNCIONARIOS 100
 #define MAX_ESTADIAS 100
 #define MAX_QUARTOS 50
 #define VALOR_DIARIA 500.0

 // Definir estruturas e variáveis globais

 typedef struct {
    int codigo;
    char nome[100];
    char endereco[100];
    char telefone[20];
    int codigo_funcionario;
 } Cliente;
```

```
typedef struct {
    int codigo;
    char nome[50];
    char cargo[50];
    float salario;
} Funcionario;

typedef struct {
    int dia;
    int mes;
    int ano;
} Data;

typedef struct {
    Data entrada;
    Data saida;
    int codigo_cliente;
    int codigo_funcionario;
    int quantidade_hospedes;
    int numero_quarto;
    float valor_total;
} Estadia;

Cliente clientes[MAX_CLIENTES];
int countClientes = 0;
```

```
Funcionario funcionarios[MAX_FUNCIONARIOS];
int countFuncionarios = 0;

Estadia estadias[MAX_ESTADIAS];
int countEstadias = 0;

// Protótipos das funções
void cadastrar_cliente();
void cadastrar_estadia();
void cadastrar_funcionario();
void listar_clientes();
void listar_estadias();
void listar_funcionarios();
int verificar_disponibilidade_quarto(int quantidade_hospedes, int *numero_quarto);
int dataValida(int dia, int mes, int ano);
int calcular_dias(Data entrada, Data saida);
float calcular_valor_estadia(Data entrada, Data saida, float valor_diaria);
int gerar_codigo_cliente();
int gerar_codigo_funcionario();
void salvar_funcionarios_txt();
void carregar_funcionarios();
void salvar_clientes_txt();
void carregar_clientes();
void mostrarMensagemSeVazio(int quantidade, const char *mensagem);

// Implementação das funções
```

```
void mostrarMensagemSeVazio(int quantidade, const char *mensagem) {
    if (quantidade == 0) {
        printf("%s\n", mensagem);
    }
}

int dataValida(int dia, int mes, int ano) {
    if (ano < 2024) return 0;
    if (mes < 1 || mes > 12) return 0;
    if (dia < 1 || dia > 31) return 0;
    return 1;
}

int calcular_dias(Data entrada, Data saida) {
    struct tm entrada_tm = {0, 0, 14, entrada.dia, entrada.mes - 1, entrada.ano - 1900};
    struct tm saida_tm = {0, 0, 12, saida.dia, saida.mes - 1, saida.ano - 1900};
    time_t tempoEntrada = mktime(&entrada_tm);
    time_t tempoSaida = mktime(&saida_tm);
    return difftime(tempoSaida, tempoEntrada) / (60 * 60 * 24);
}

float calcular_valor_estadia(Data entrada, Data saida, float valor_diaria) {
    int dias = calcular_dias(entrada, saida);
    float valor_total = dias * valor_diaria;
    return valor_total;
```

```
}

int verificar_disponibilidade_quarto(int quantidade_hospedes, int *numero_quarto) {
    *numero_quarto = rand() % MAX_QUARTOS + 1;
    return 1;
}

int gerar_codigo_cliente() {
    return rand() % 10000 + 1; // Gerar um código aleatório entre 1 e 10000
}

int gerar_codigo_funcionario() {
    return rand() % 1000 + 1; // Gerar um código aleatório entre 1 e 1000
}

void cadastrar_cliente() {
    if (countClientes >= MAX_CLIENTES) {
        printf("Limite máximo de clientes atingido.\n");
        return;
    }

    getchar();
    printf("Digite o nome completo do cliente: ");
    fgets(clientes[countClientes].nome, sizeof(clientes[countClientes].nome), stdin);
    clientes[countClientes].nome[strcspn(clientes[countClientes].nome, "\n")] = '\0';
```

```
    printf("Digite o endereço do cliente: ");
    fgets(clientes[countClientes].endereco, sizeof(clientes[countClientes].endereco), stdin);
    clientes[countClientes].endereco[strcspn(clientes[countClientes].endereco, "\n")] = '\0';

    while (1) {
        printf("Digite o telefone (apenas 9 dígitos) do cliente: ");
        fgets(clientes[countClientes].telefone, sizeof(clientes[countClientes].telefone), stdin);
        clientes[countClientes].telefone[strcspn(clientes[countClientes].telefone, "\n")] = '\0';

        if (strlen(clientes[countClientes].telefone) != 9) {
            printf("Telefone inválido. Deve conter exatamente nove dígitos.\n");
        } else {
            break;
        }
    }

    clientes[countClientes].codigo = gerar_codigo_cliente();
    clientes[countClientes].codigo_funcionario = gerar_codigo_funcionario(); // Atribuir um funcionário responsável (simulação)

    printf("Cliente cadastrado com sucesso. Código do cliente: %d\n", clientes[countClientes].codigo);
    countClientes++;

    salvar_clientes_txt(); // Salvar clientes no arquivo após cada cadastro
}
```

```
void cadastrar_estadia() {
    int codigo_cliente;
    int quantidade_hospedes;
    Data entrada, saida;

    printf("\n=== Cadastro de Estadia ===\n");
    printf("\nValor da Diária: R$500.00\n");

    if (countClientes == 0) {
        printf("\nNão há clientes cadastrados. Cadastre um cliente primeiro.\n");
        return;
    }

    printf("\nDigite o código do cliente que deseja se hospedar: ");
    scanf("%d", &codigo_cliente);
    getchar();

    Cliente *cliente = NULL;
    for (int i = 0; i < countClientes; i++) {
        if (clientes[i].codigo == codigo_cliente) {
            cliente = &clientes[i];
            break;
        }
    }

    if (cliente == NULL) {
```

```
    printf("Cliente não encontrado. Verifique o código e tente novamente.\n");
    return;
}

printf("Digite a quantidade de hóspedes: ");
scanf("%d", &quantidade_hospedes);
getchar();

while (1) {
    printf("Digite a data de entrada desejada (DD/MM/AAAA): ");
    scanf("%d/%d/%d", &entrada.dia, &entrada.mes, &entrada.ano);
    getchar();

    if (!dataValida(entrada.dia, entrada.mes, entrada.ano)) {
        printf("Data de entrada inválida. Digite novamente.\n");
        continue;
    }

    printf("Digite a data de saída desejada (DD/MM/AAAA): ");
    scanf("%d/%d/%d", &saida.dia, &saida.mes, &saida.ano);
    getchar();

    if (!dataValida(saida.dia, saida.mes, saida.ano)) {
        printf("Data de saída inválida. Digite novamente.\n");
        continue;
    }
```

```
        if (saida.ano < entrada.ano ||
            (saida.ano == entrada.ano && saida.mes < entrada.mes) ||
            (saida.ano == entrada.ano && saida.mes == entrada.mes && saida.dia <=
entrada.dia)) {
            printf("Erro: Data de saída deve ser posterior à data de entrada. Digite novamente.
\n");
            continue;
        }

        int numero_quarto;
        if (!verificar_disponibilidade_quarto(quantidade_hospedes, &numero_quarto)) {
            printf("Quarto não disponível para as datas selecionadas. Escolha outras datas.\n");
            continue;
        }

        float valor_total_estadia = calcular_valor_estadia(entrada, saida, VALOR_DIARIA);

        estadias[countEstadias].entrada = entrada;
        estadias[countEstadias].saida = saida;
        estadias[countEstadias].codigo_cliente = cliente->codigo;
        estadias[countEstadias].codigo_funcionario = cliente->codigo_funcionario;
        estadias[countEstadias].quantidade_hospedes = quantidade_hospedes;
        estadias[countEstadias].numero_quarto = numero_quarto;
        estadias[countEstadias].valor_total = valor_total_estadia;

        countEstadias++;
```

```
        printf("Estadia cadastrada com sucesso.\n");
        break;
    }
 }

void cadastrar_funcionario() {
   while (1) {
      if (countFuncionarios >= MAX_FUNCIONARIOS) {
         printf("Limite máximo de funcionários atingido.\n");
         return;
      }

      getchar();
      printf("Digite o nome completo do funcionário: ");
      fgets(funcionarios[countFuncionarios].nome,
sizeof(funcionarios[countFuncionarios].nome), stdin);
      funcionarios[countFuncionarios].nome[strcspn(funcionarios[countFuncionarios].nome,
"\n")] = '\0';

      char cargo[50];
      int cargo_valido = 0;
      while (!cargo_valido) {
         printf("Digite o cargo do funcionário (recepcionista, auxiliar de limpeza, garçom,
gerente): ");
         fgets(cargo, sizeof(cargo), stdin);
         cargo[strcspn(cargo, "\n")] = '\0';
```

```
        // Verifica se o cargo é válido
        if (strcmp(cargo, "recepcionista") == 0 ||
            strcmp(cargo, "auxiliar de limpeza") == 0 ||
            strcmp(cargo, "garçom") == 0 ||
            strcmp(cargo, "gerente") == 0) {
            cargo_valido = 1;
        } else {
            printf("Cargo inválido. Escolha entre recepcionista, auxiliar de limpeza, garçom ou gerente.\n");
        }
    }

    strcpy(funcionarios[countFuncionarios].cargo, cargo);

    printf("Digite o salário do funcionário: ");
    scanf("%f", &funcionarios[countFuncionarios].salario);
    getchar();

    funcionarios[countFuncionarios].codigo = gerar_codigo_funcionario();

    printf("Funcionário cadastrado com sucesso. Código do funcionário: %d\n", funcionarios[countFuncionarios].codigo);
    countFuncionarios++;

    salvar_funcionarios_txt(); // Salvar funcionários no arquivo após cada cadastro
```

```
        char opcao;
        printf("Deseja cadastrar outro funcionário? (s/n): ");
        scanf(" %c", &opcao);
        getchar();

        if (opcao != 's' && opcao != 'S') {
            break; // Sai do loop se a opção não for 's' ou 'S'
        }
    }
}

 void listar_clientes() {
    printf("\n=== Lista de Clientes ===\n");
    mostrarMensagemSeVazio(countClientes, "\nNão há clientes cadastrados.");

    for (int i = 0; i < countClientes; i++) {
        printf("\nCódigo: %d\n", clientes[i].codigo);
        printf("Nome: %s\n", clientes[i].nome);
        printf("Endereço: %s\n", clientes[i].endereco);
        printf("Telefone: %s\n", clientes[i].telefone);
        printf("Código Funcionário: %d\n", clientes[i].codigo_funcionario);
        printf("--------------------\n");
    }
 }

 void listar_estadias() {
```

```
    printf("\n=== Lista de Estadias ===\n");
    mostrarMensagemSeVazio(countEstadias, "\nNão há estadias cadastradas.");

    for (int i = 0; i < countEstadias; i++) {
        printf("\nCódigo Cliente: %d\n", estadias[i].codigo_cliente);
        printf("Código Funcionário: %d\n", estadias[i].codigo_funcionario);
        printf("Data de Entrada: %02d/%02d/%04d\n", estadias[i].entrada.dia,
estadias[i].entrada.mes, estadias[i].entrada.ano);
        printf("Data de Saída: %02d/%02d/%04d\n", estadias[i].saida.dia,
estadias[i].saida.mes, estadias[i].saida.ano);
        printf("Número do Quarto: %d\n", estadias[i].numero_quarto);
        printf("Quantidade de Hóspedes: %d\n", estadias[i].quantidade_hospedes);
        printf("Valor Total: R$ %.2f\n", estadias[i].valor_total);
        printf("--------------------\n");
    }
 }

 void listar_funcionarios() {
    printf("\n=== Lista de Funcionários ===\n");
    mostrarMensagemSeVazio(countFuncionarios, "\nNão há funcionários cadastrados.\n");

    for (int i = 0; i < countFuncionarios; i++) {
        printf("\nCódigo: %d\n", funcionarios[i].codigo);
        printf("Nome: %s\n", funcionarios[i].nome);
        printf("Cargo: %s\n", funcionarios[i].cargo);
        printf("Salário: R$ %.2f\n", funcionarios[i].salario);
        printf("--------------------\n");
```

```
    }
}

void salvar_clientes_txt() {
    FILE *arquivo = fopen("clientes.txt", "w");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo clientes.txt");
        return;
    }

    for (int i = 0; i < countClientes; i++) {
        fprintf(arquivo, "%d;%s;%s;%s;%d\n", clientes[i].codigo, clientes[i].nome,
clientes[i].endereco, clientes[i].telefone, clientes[i].codigo_funcionario);
    }

    fclose(arquivo);
}

void salvar_funcionarios_txt() {
    FILE *arquivo = fopen("funcionarios.txt", "w");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo funcionarios.txt");
        return;
    }

    for (int i = 0; i < countFuncionarios; i++) {
        fprintf(arquivo, "%d;%s;%s;%.2f\n", funcionarios[i].codigo, funcionarios[i].nome,
```

```
funcionarios[i].cargo, funcionarios[i].salario);
    }

    fclose(arquivo);
 }

 void carregar_clientes() {
    FILE *arquivo = fopen("clientes.txt", "r");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo clientes.txt");
        return;
    }

    while (fscanf(arquivo, "%d;%[^;];%[^;];%[^;];%d\n", &clientes[countClientes].codigo,
clientes[countClientes].nome, clientes[countClientes].endereco,
clientes[countClientes].telefone, &clientes[countClientes].codigo_funcionario) == 5) {
        countClientes++;
    }

    fclose(arquivo);
 }

 void carregar_funcionarios() {
    FILE *arquivo = fopen("funcionarios.txt", "r");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo funcionarios.txt");
        return;
```

```
    }

    while (fscanf(arquivo, "%d;%[^;];%[^;];%f\n", &funcionarios[countFuncionarios].codigo,
funcionarios[countFuncionarios].nome, funcionarios[countFuncionarios].cargo,
&funcionarios[countFuncionarios].salario) == 4) {
        countFuncionarios++;
    }

    fclose(arquivo);
}

int main() {
    srand(time(NULL)); // Inicializar gerador de números aleatórios com o tempo atual

    carregar_clientes();
    carregar_funcionarios();

    int opcao;
    do {
        printf("\nSeja Bem Vindo ao Hotel Descanso Garantido!\n");
        printf("\n=== Menu Principal ===\n");
        printf("\n1 - Cadastrar Cliente\n");
        printf("2 - Cadastrar Estadia\n");
        printf("3 - Cadastrar Funcionário\n");
        printf("4 - Listar Clientes\n");
        printf("5 - Listar Estadias\n");
        printf("6 - Listar Funcionários\n");
```

```
printf("7 - Sair\n");
printf("\nEscolha uma opção: ");
scanf("%d", &opcao);
getchar(); // Limpar o buffer de entrada

switch (opcao) {
    case 1:
        cadastrar_cliente();
        break;
    case 2:
        cadastrar_estadia();
        break;
    case 3:
        cadastrar_funcionario();
        break;
    case 4:
        listar_clientes();
        break;
    case 5:
        listar_estadias();
        break;
    case 6:
        listar_funcionarios();
        break;
    case 7:
        printf("Encerrando o programa...\n");
```

```
            break;
        default:
            printf("Opção inválida. Tente novamente.\n");
    }
} while (opcao != 7);

return 0;
}
```