PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
UNIDADE EDUCACIONAL CORAÇÃO EUCARISTICO
Bacharelado em Engenharia de Software

Integrantes: Bernardo VInhal de Carvalho e Douglas Marçal de Freitas
Projeto: LocHoteis

Apresentação:

O projeto visa o controle organizacional nas dependências do hotel “Descanso garantido”, fazendo o controle de estadias, clientes e funcionários, com funções para:

- Cadastrar os clientes.
- Cadastrar os funcionários.
- Cadastrar e ver o status das estadias do hotel.

Desenvolvidas em 3 sprints e o processo de Scrum para gerenciar o progresso.

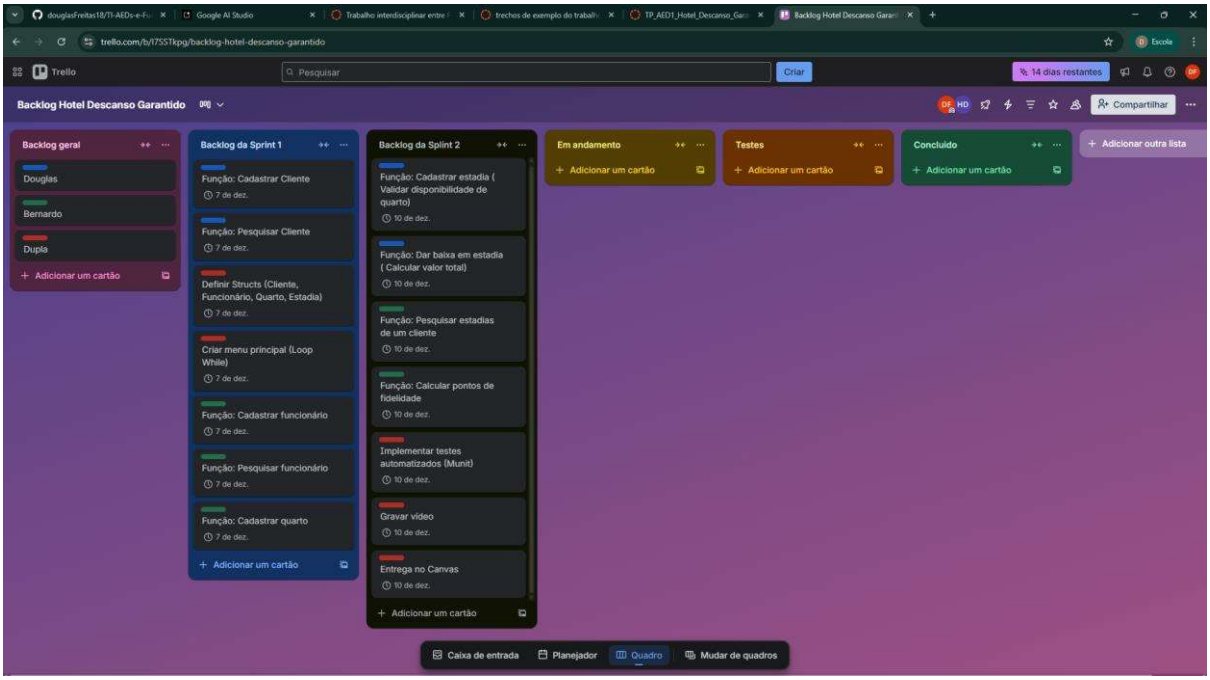
Backlog do produto:

Fizemos a organização no Trello com divisões dos quadros e dividiram o processo em 4 (Quatro) etapas, foram elas

1.1 - Backlog inicial
1.2 - Sprint 1
1.3 - Sprint 2
1.4 - Estado final do backlog

Segue o planejamento inicial das atividades:

Figura 1 – Quadro e divisão de tarefas

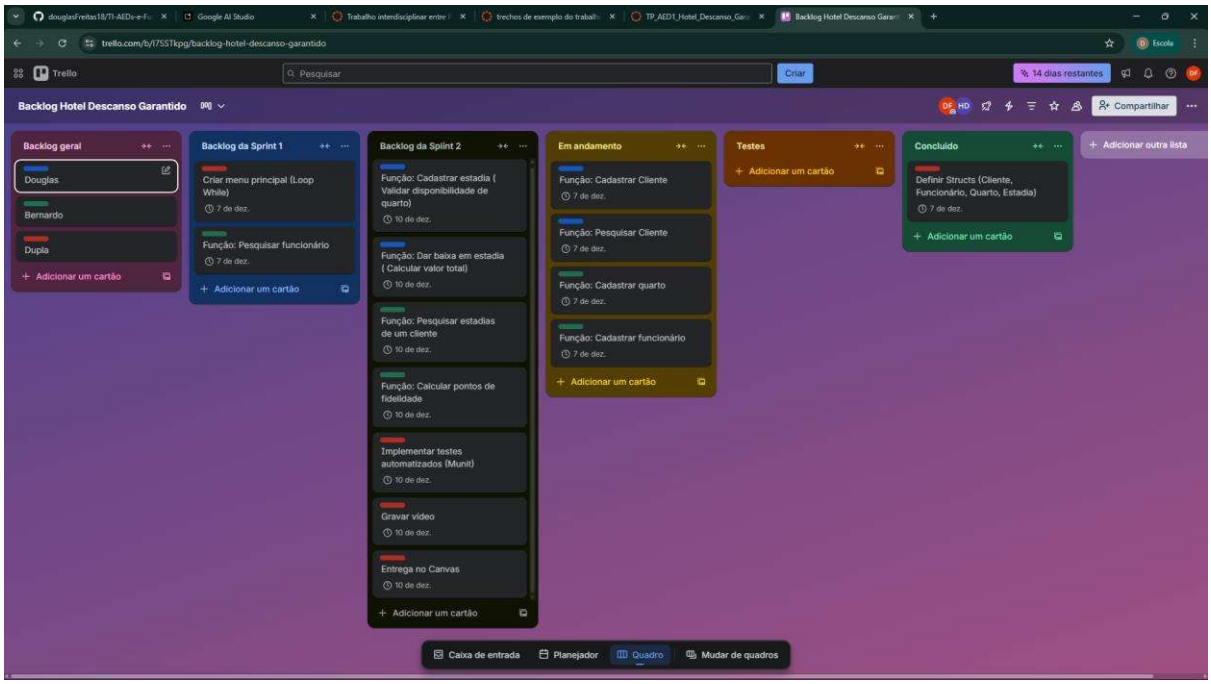


1.1 - Backlog inicial

No início do projeto, todas as funcionalidades solicitadas na especificação foram mapeadas e inseridas na coluna "Backlog Geral". Nenhuma tarefa havia sido iniciada. A equipe se reuniu para analisar os requisitos e definir as prioridades.

Itens:
- Criar um menu principal
- Implementar CRUD de Clientes (Cadastrar e pesquisar)
- Definir Estruturas de Dados (Structs) e Arquivos.
- Implementar CRUD de Funcionários (Cadastrar e Pesquisar)
- Implementar Cadastro de Quartos
- Implementar Cadastro de Estadias(Com validação de datas)
- Implementar Baixa de estadias(Cálculo de valor)
- Implementar Cálculo de pontos de fidelidade
- Testes Automatizados (Munit)
- Gravação de vídeo

Figura 1.1 – Backlog inicial



Fonte: Elaborado pelo autor

1.2 - Sprint 1

Criar a base do sistema, permitindo a persistência de dados em arquivos binários e a navegação pelo menu.

Modificações no Trello:
As tarefas de infraestrutura e cadastros simples foram movidas para **"Em Andamento"** e, posteriormente, para **"Concluído"**.

Tarefa: Definir Structs (Cliente, Funcionário, Quarto, Estadia) e constantes.
Responsável: Dupla
Status: Concluído

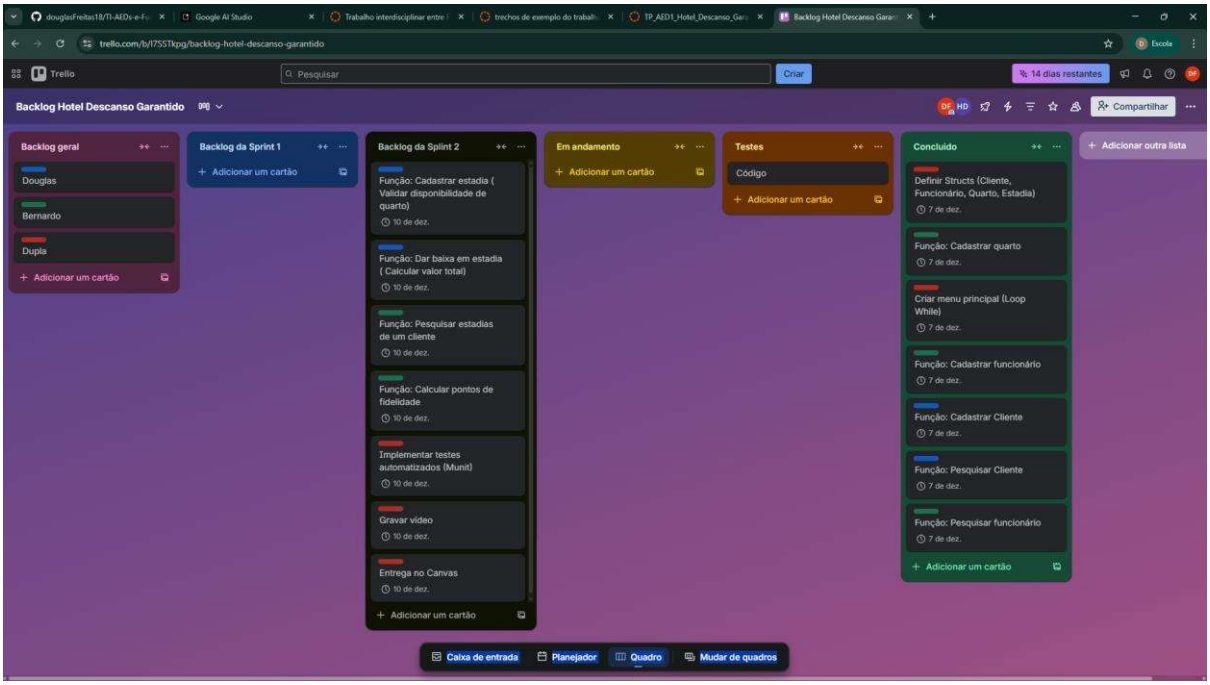
Tarefa: Criar Menu Principal (Loop `main.c`).
Responsável: Dupla
Status: Concluído

Tarefa: Módulo de Clientes (Cadastrar/Pesquisar).
Responsável: Douglas
Status: Concluído

Tarefa: Módulo de Funcionários (Cadastrar/Pesquisar) e Quartos
Responsável: Bernardo
Status: Concluído

Resultado da Sprint: O software já compilava, exibia o menu e salvava/lia dados básicos (Clientes, Funcionários e Quartos) na pasta dados/.

Figura 1.2 – Final primeira sprint menu e inicialização de dados (Sprint 1)



Fonte: Elaborado pelo autor

1.3 - Sprint 2

Implementar as regras de negócio complexas (datas, disponibilidade), realizar testes e refatorar o código para entrega.
O foco mudou para as tarefas restantes no Backlog Geral. Foi adicionada uma tarefa extra de **"Refatoração e Modularização"** para melhorar a qualidade do código final.

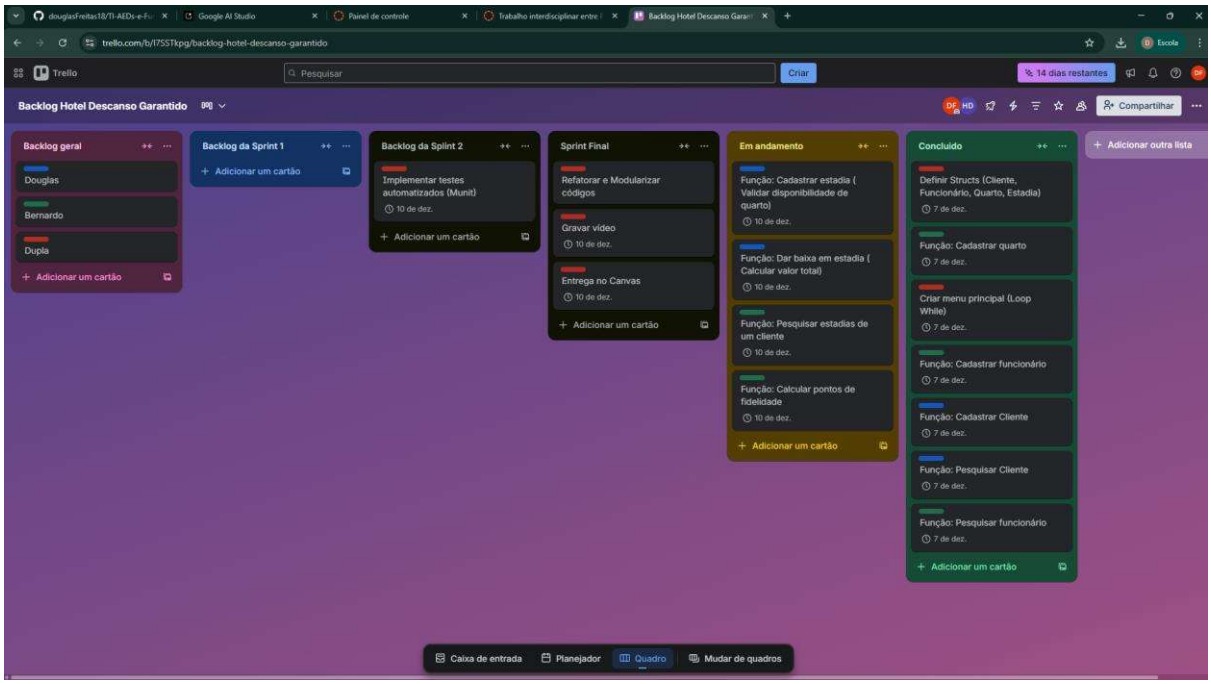
Tarefa: Cadastrar Estadia (Verificação de disponibilidade e colisão de datas).
Responsável: Bernardo.
Status: Concluído (Movido para o Backlog -> Concluido)

Tarefa: Implementar Testes Automatizados (Biblioteca Munit).
Responsável: Dupla.
Status: Concluído.

Tarefa (Nova): Refatoração e Modularização (Divisão em `clientes.c`, `quartos.c`, etc.).
Responsável: Dupla.
Status: Concluído.

Tarefa: Gravação do Vídeo e Documentação.
Responsável: Dupla.
Status: Concluído.

Figura 1.3 – Final da segunda sprint e encerramento



Fonte: Elaborado pelo autor

1.4 - Estado final do backlog

Ao final do projeto (data de entrega), a coluna **"Backlog Geral"** encontra-se vazia e todas as tarefas obrigatórias encontram-se na coluna **"Concluído"**. Isso demonstra que o escopo do projeto foi 100% atendido dentro do prazo estipulado.

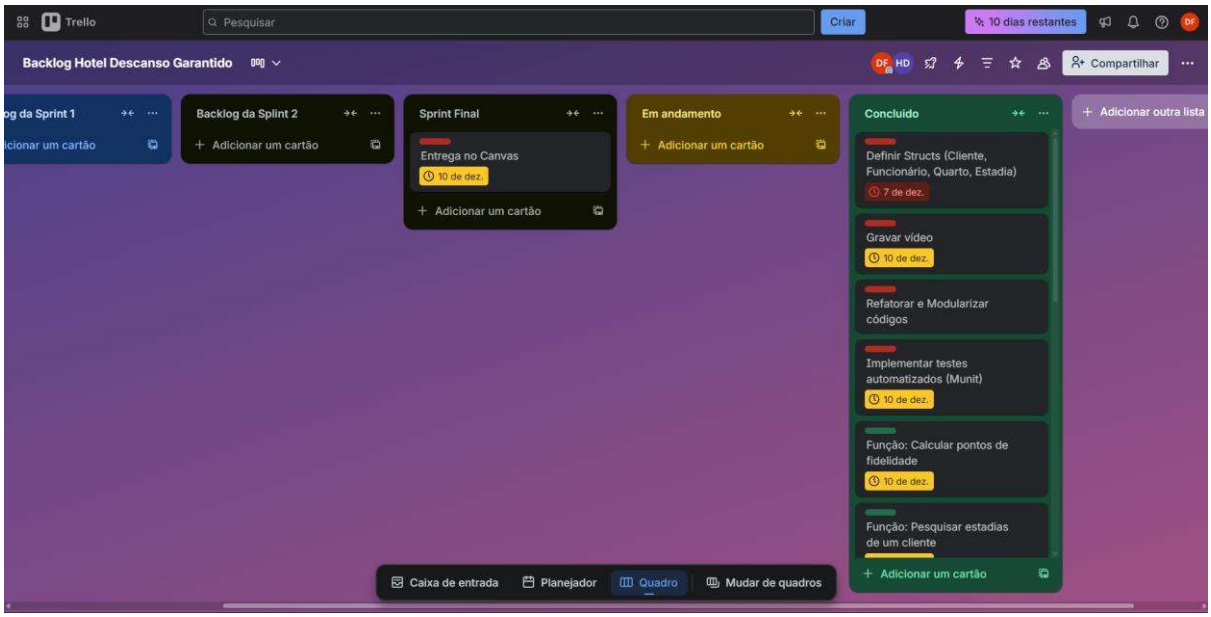
Resumo da divisão:

Douglas: Focou na gestão de Clientes e no ciclo financeiro da estadia (Baixa/Fidelidade).

- **Bernardo:** Focou na gestão de Recursos (Funcionários/Quartos) e na lógica de alocação (Reserva/Datas).

Conjunto: Arquitetura inicial, Menu, Testes Automatizados e Refatoração.

Figura 1.4 - Entrega do produto final



Fonte: Elaborado pelo autor

Lista de assinaturas das funções e parâmetros

Abaixo estão as Explicações da estrutura de dados principal do programa e apresentação das assinaturas das funções

- int calcular_dias(Data entrada, Data saida)
Função responsável por calcular a duração da estadia. Recebe como parâmetros duas estruturas do tipo `Data` (entrada e saída). Converte as datas para segundos (usando `time_t`) e retorna a diferença em dias (número inteiro).
- Retorna um inteiro positivo representando a quantidade de diárias.
- Retorna valor negativo ou zero caso a data de saída seja anterior à de entrada.

- int verifica_sobreposicao_datas(Data nova_ent, Data nova_sai, Data antiga_ent, Data antiga_sai)
Função crítica para verificar conflitos de agenda. Recebe quatro parâmetros do tipo `Data`: o intervalo desejado (nova entrada/saída) e um intervalo já ocupado (antiga entrada/saída). Verifica matematicamente se os intervalos de tempo se cruzam.
- Retorna **1**: Se houver colisão de datas (o quarto está ocupado nesse período).
- Retorna **0**: Se as datas não colidem (o quarto está livre).

- time_t converter_para_time(Data d)

Função auxiliar que recebe uma estrutura `Data` e a converte para o tipo `time_t` da biblioteca padrão `<time.h>`. Utilizada internamente pela função `calcular_dias` para permitir operações aritméticas com datas

Módulo de clientes

int verificar_cliente_existe(int codigo)

Função de validação que recebe o `codigo` (inteiro) de um cliente. Percorre o arquivo binário de clientes buscando por este código
Retorna **1**: Se o cliente já existe.
Retorna **0**: Se o cliente não foi encontrado.

- void cadastrar_cliente()

Procedimento para registro de novos clientes. Não recebe parâmetros, pois coleta os dados (código, nome, endereço, telefone) diretamente da entrada padrão (teclado) e os persiste no arquivo `clientes.bin` após validar a unicidade do código

- void pesquisar_cliente()

Procedimento de busca. Solicita ao usuário um código via teclado, varre o arquivo de clientes e imprime na tela os dados do registro encontrado, ou uma mensagem de erro caso não exista.

Módulo de funcionários e Quartos

- void cadastrar_funcionario()

Procedimento para registrar funcionários. Coleta dados (código, nome, cargo, salário) e salva no arquivo `funcionarios.bin`. Garante que não existam dois funcionários com o mesmo código

- void cadastrar_quarto()

Procedimento para registrar quartos. Coleta o número, capacidade e valor da diária. Inicializa o status do quarto como "0" (Desocupado) e salva no arquivo `quartos.bin`

- double buscar_valor_diaria(int num_quarto)

Função auxiliar utilizada no momento do checkout. Recebe o `num_quarto` e busca no arquivo de quartos
- Retorna o valor da diária (`double`) se o quarto for encontrado o valor correspondente à diária.
- Retorna **0.0** caso o quarto não seja encontrado.

Modulos de estadias

- int encontrar_quarto_disponivel(int qtd_hospedes, Data ent, Data sai)

Função complexa que busca um quarto vago. Recebe a capacidade desejada (`qtd_hospedes`) e o intervalo de datas (`ent`, `sai`). Percorre todos os quartos e, para cada quarto com capacidade suficiente, verifica todas as estadias ativas para garantir que não haja choque de horário usando a função `verifica_sobreposicao_datas`
- Retorna o **número do quarto** (inteiro) se encontrar um disponível.
- Retorna **-1** se não houver quartos disponíveis para os critérios informados.

- void cadastrar_estadia()
Procedimento principal de reserva. Solicita código do cliente e datas. Chama internamente `encontrar_quarto_disponivel`. Se houver vaga, grava uma nova estrutura `Estadia` no arquivo com status ativo (1). Caso contrário, informa o usuário sobre a indisponibilidade.

- void dar_baixa_estadia()
Procedimento de checkout. Solicita o código da estadia, verifica se ela está ativa, calcula o valor total a pagar (dias * valor da diária) e exibe na tela. Atualiza o status da estadia no arquivo para finalizada (0). Exibe também os pontos de fidelidade adquiridos
Procedimento de checkout. Solicita o código da estadia, verifica se ela está ativa, calcula o valor total a pagar (dias * valor da diária) e exibe na tela. Atualiza o status da estadia no arquivo para finalizada (0). Exibe também os pontos de fidelidade adquiridos

- void calcular_pontos_fidelidade()
Procedimento de relatório. Solicita o código do cliente e percorre todo o histórico de estadias (ativas e finalizadas) daquele cliente, somando 10 pontos para cada diária consumida. Exibe o saldo total de pontos na tela.

Casos de teste do software

Os testes foram planejados para cobrir as validações críticas do sistema, garantindo a integridade dos arquivos binários e a lógica de negócios (datas e cálculos financeiros)

Tabela 1 – Testes do módulo de cadastro (Clientes;Funcionários;Quartos)

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Código do Cliente	Código inteiro positivo inédito (ex: 10, 20).	O sistema aceita o código e prossegue para solicitar o Nome.	Código já existente no banco de dados (ex: tentar cadastrar o código 10 novamente).	O sistema exibe a mensagem "ERRO: Já existe um cliente com o código X" e cancela a operação.
Número do Quarto	Número inteiro inédito (ex: 101, 102).	O quarto é cadastrado com status "Desocupado".	Número de quarto já existente.	Mensagem de erro informando duplicidade e retornando ao menu.
Capacidade/Hóspedes	Número inteiro maior que 0.	Capacidade registrada corretamente.	(Não aplicável pois o sistema aceita, mas lógica de negócio exige > 0).	N/A

Tabela 2 – Testes do módulo de lógica de datas (Estadias)

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Datas de Entrada e Saída	Data de Saída posterior à Data de Entrada (ex: Ent: 01/01/2024, Sai: 05/01/2024).	O sistema calcula a diferença de dias (4 dias) e prossegue para buscar quartos.	Data de Saída anterior ou igual à Entrada (ex: Ent: 05/01, Sai: 01/01).	Mensagem "Erro: Data de saída deve ser depois da entrada" e cancelamento da reserva.
Disponibilidade de Quarto	Intervalo de datas solicitado não coincide com nenhuma estadia *ativa* naquele quarto.	O sistema encontra o quarto disponível, calcula o valor e gera o Código da Estadia.	Intervalo de datas coincide (total ou parcialmente) com uma estadia já cadastrada para aquele quarto.	Mensagem "Desculpe, não há quartos disponíveis para essa capacidade nessas datas".

abela 3 – Testes do Módulo Financeiro (Baixa e Fidelidade)

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Código da Estadia (Baixa)	Código de uma estadia existente e com status "Ativa" (1).	O sistema calcula: (Dias x Valor Diária), exibe o total na tela e altera o status para "Finalizada".	Código inexistente OU Código de uma estadia já finalizada.	Mensagem "Estadia não encontrada" ou "Essa estadia já foi finalizada!".
Cálculo de Fidelidade	Código de cliente com estadias no histórico.	Exibe a soma de todas as diárias multiplicada por 10.	Código de cliente sem histórico.	Exibe "Total de 0 pontos".

Testes Automatizados (Munit)

Como complemento à validação manual, foram implementados testes unitários automatizados utilizando a biblioteca **µnit (munit)** para validar a lógica matemática do sistema isoladamente:

Cálculo de Dias: Verificação da função `difftime` entre datas.
Sobreposição de Datas: Simulação de 3 cenários (interseção, antes e depois) para garantir que a lógica de `verifica_sobreposicao_datas` está perfeita.
Persistência: Teste de criação e leitura de arquivo binário controlado.

Resultado esperado:
Todos os testes passaram com êxito (OK), garantindo a robustez das funções principais antes da integração com o menu do usuário.

O código C do programa e suas funções incluindo a implementação automatizada dos
casos de testes.