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

Flávio de Souza Ferreira Junior

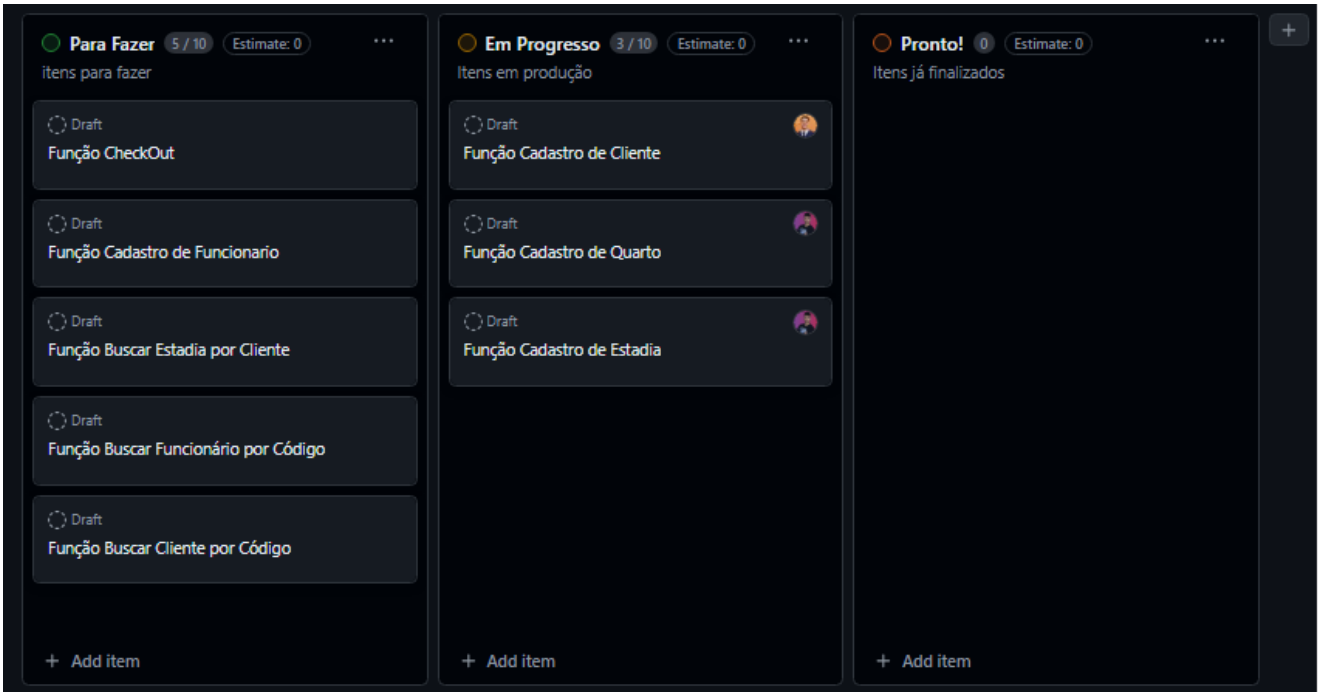
Bernardo Resende

Apresentação:

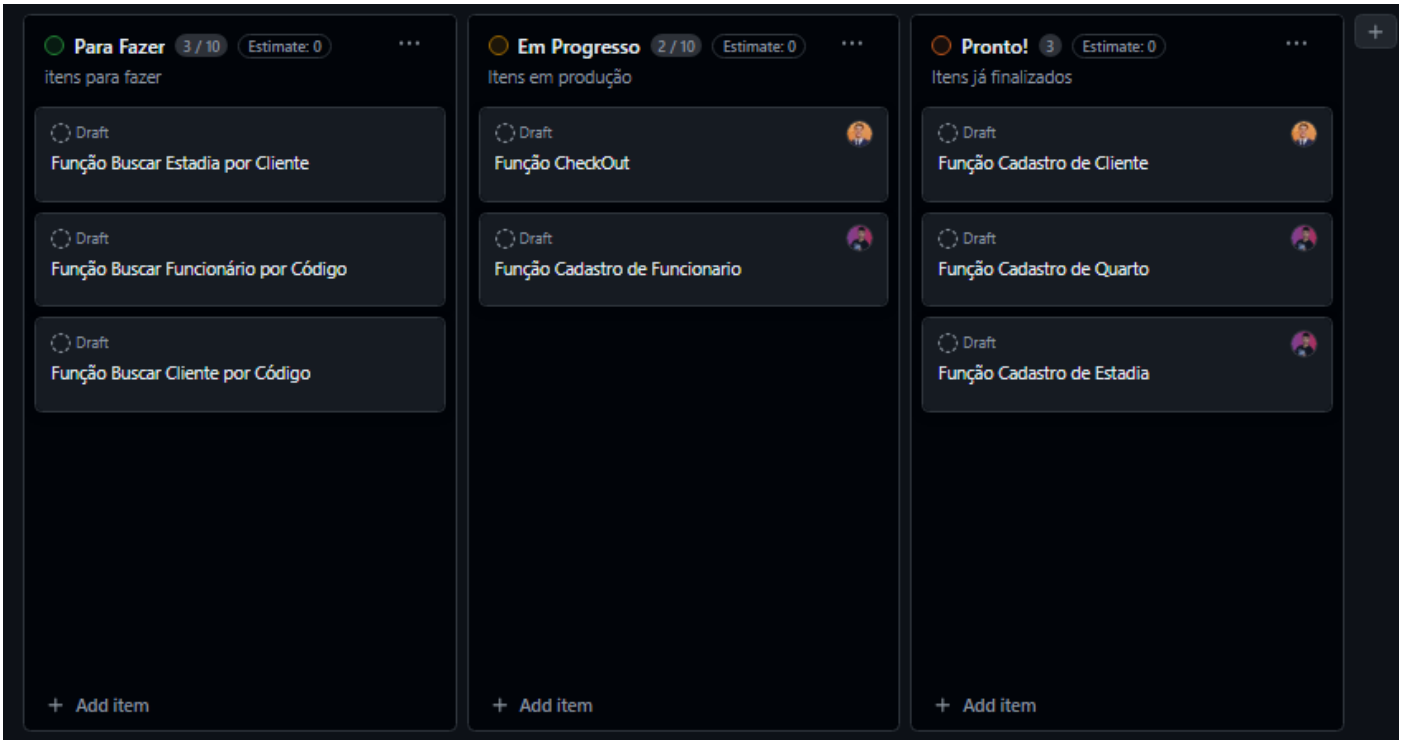
O Hotel Descanso Garantido recebe milhares de hóspedes todos os meses, com essa demanda, eles nos procuraram pra desenvolver um software que gerencie todos os seus processos, afim de facilitar a organização e ter maior eficiência em seus serviços.

Backlog do produto:

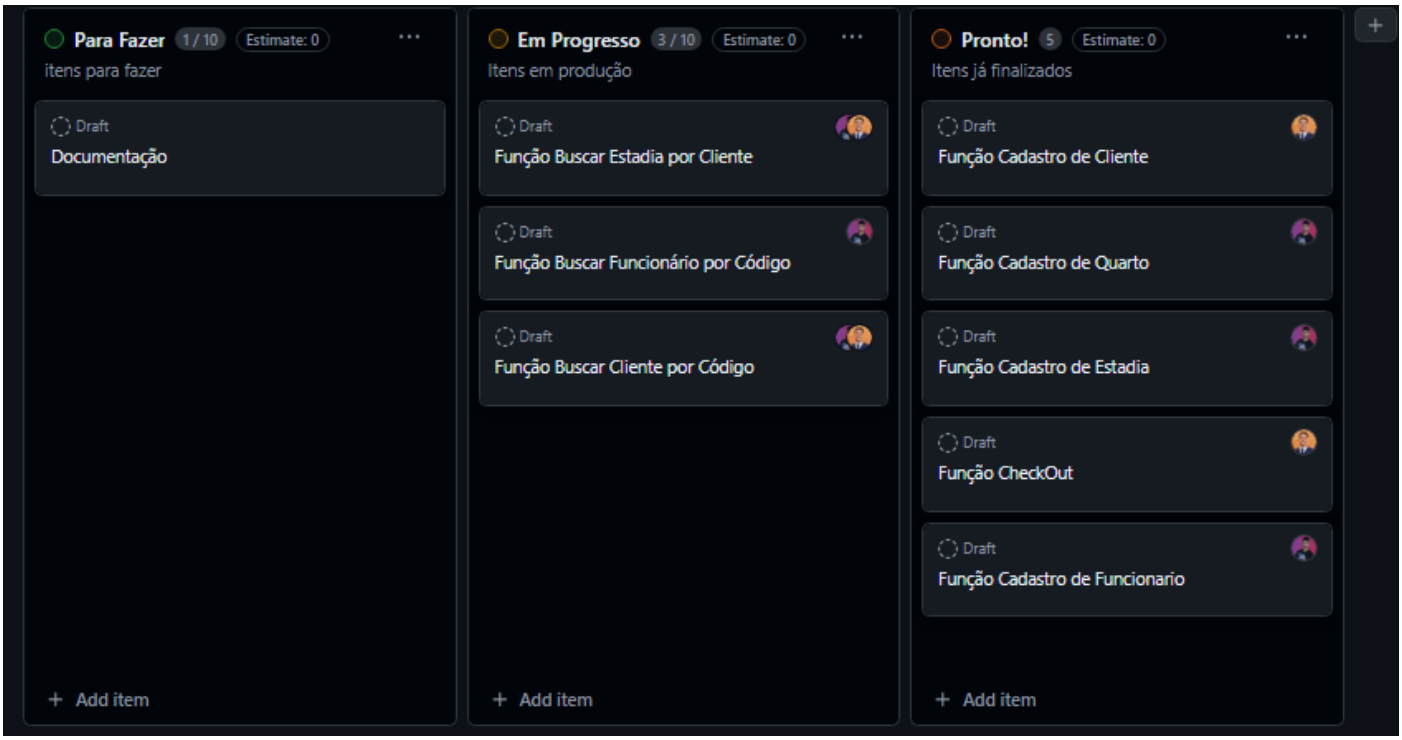
SPRINT 1: (2 DIAS)



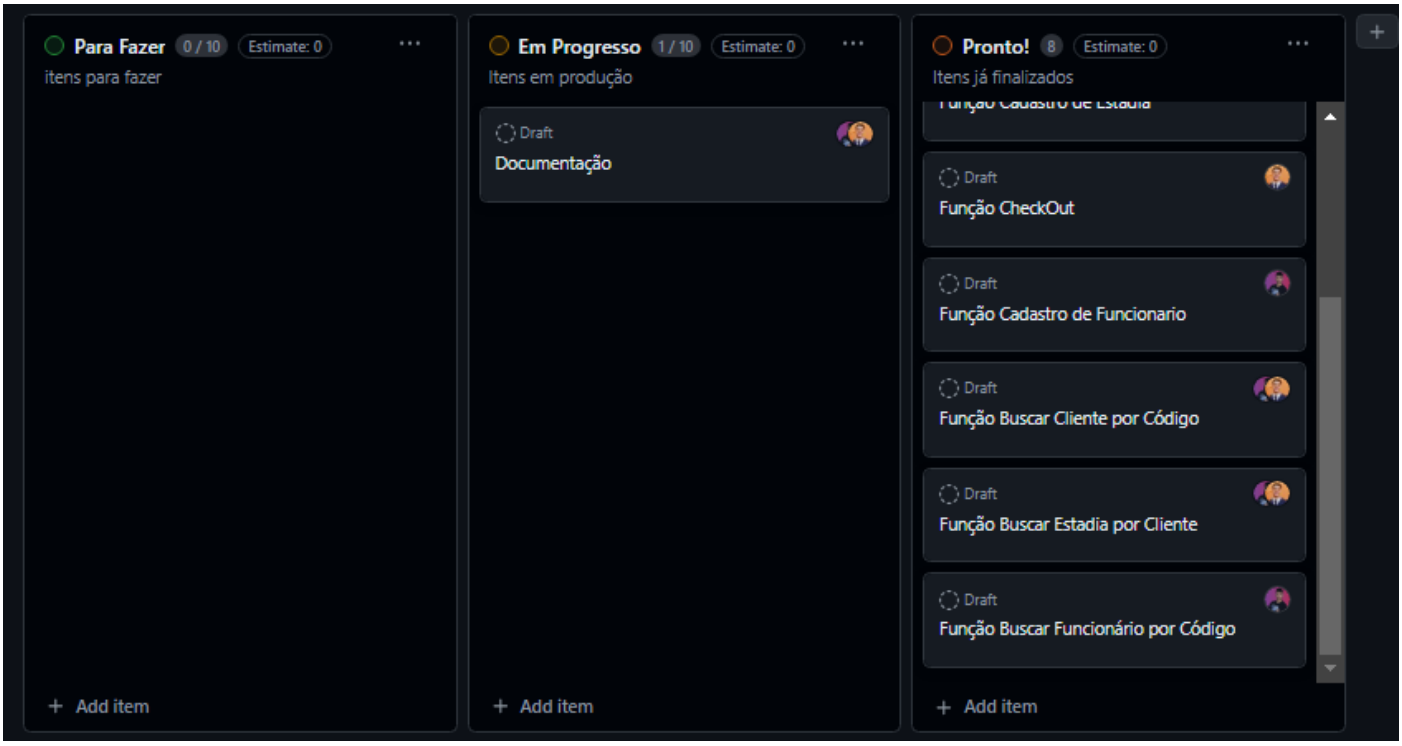
SPRINT 2: (1 DIAS)



SPRINT 3: (2 DIAS)



SPRINT 4: (2 DIAS)



Lista de assinaturas das funções e parâmetros:

Função `gerarCodigo`

Descrição:

A função `gerarCodigo` é responsável por gerar e retornar códigos únicos para diferentes tipos de entidades dentro de um sistema. Os tipos suportados são "clientes", "funcionarios", "quartos" e "estadias". A função incrementa e retorna o próximo código disponível para cada tipo especificado.

Assinatura:

int gerarCodigo(const char *tipo);

Parâmetros:

- `tipo`: Um ponteiro para uma string (`const char *`) que especifica o tipo de entidade para a qual um código deve ser gerado ("clientes", "funcionarios", "quartos" ou "estadias").

Retorno:

- Retorna um valor inteiro:

- O próximo código disponível para o tipo especificado.

- `-1` se o tipo fornecido não corresponder a nenhum dos tipos suportados ("clientes", "funcionarios", "quartos" ou "estadias").

Variáveis Globais Utilizadas:

- `proximoCodigoCliente`: Variável global que armazena o próximo código disponível para clientes.

- `proximoCodigoFuncionario`: Variável global que armazena o próximo código disponível para funcionários.

- `proximoNumeroQuarto`: Variável global que armazena o próximo número disponível para quartos.

- `proximoCodigoEstadia`: Variável global que armazena o próximo código disponível para estadias.

Comportamento:

A função realiza as seguintes operações:

1. Compara o valor da string `tipo` com os tipos suportados usando `strcmp`.

2. Incrementa a variável global correspondente ao tipo encontrado.

3. Retorna o valor atualizado da variável global correspondente.

4. Retorna `-1` se o tipo fornecido não estiver entre os tipos suportados.

Exemplo de Uso:

int codigoCliente = gerarCodigo("clientes");

int codigoFuncionario = gerarCodigo("funcionarios");

int numeroQuarto = gerarCodigo("quartos");

int codigoEstadia = gerarCodigo("estadias");

- Certifique-se de que as variáveis globais `proximoCodigoCliente`, `proximoCodigoFuncionario`, `proximoNumeroQuarto` e `proximoCodigoEstadia` sejam inicializadas corretamente antes de chamar a função `gerarCodigo`.

- O tipo fornecido deve ser uma string válida entre "clientes", "funcionarios", "quartos" e "estadias" para garantir o comportamento esperado da função.

Esse formato foi projetado para ser facilmente copiado para o Word ou qualquer outro editor de texto, mantendo a formatação clara e organizada da documentação da função `gerarCodigo`.

Função `calcularDiarias`

Descrição:

Calcula o número de diárias entre duas datas fornecidas no formato "dia/mês/ano".

Assinatura:

int calcularDiarias(const char *dataEntrada, const char *dataSaida);

Parâmetros:

- `dataEntrada`: Uma string contendo a data de entrada no formato "dia/mês/ano".

- `dataSaida`: Uma string contendo a data de saída no formato "dia/mês/ano".

Retorno:

- Retorna um valor inteiro representando o número de dias entre `dataEntrada` e `dataSaida`. Se `dataSaida` for anterior a `dataEntrada`, o valor retornado será negativo.

Comportamento:

1. Converte as datas fornecidas em números inteiros (`dia`, `mês`, `ano`).

2. Calcula o número de dias desde o início do calendário gregoriano, utilizando uma simplificação.

3. Calcula a diferença entre os dias convertidos para obter o número de diárias entre as duas datas.

4. Retorna esse valor como o número de diárias.

Exemplo de Uso:

int numeroDiarias = calcularDiarias("10/06/2024", "15/06/2024");

// Retorna 5 (5 dias de diferença entre 10/06/2024 e 15/06/2024)

Notas:

- As datas fornecidas devem estar no formato "dia/mês/ano" para que a função funcione corretamente.

- O cálculo considera um método simplificado para converter datas em dias desde o início do calendário gregoriano, o que pode não ser totalmente preciso em contextos que exigem precisão extrema.

Função `cadastrarCliente`

Descrição:

Realiza o cadastro de um novo cliente, coletando informações como nome, endereço e telefone, e registrando esses dados em um arquivo "clientes.txt".

Assinatura:

void cadastrarCliente();

Comportamento:

1. Abre o arquivo "clientes.txt" no modo de adição (`"a"`). Se o arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e retorna.

2. Obtém o próximo código disponível para clientes chamando a função `gerarCodigo("clientes")`.

3. Solicita ao usuário que insira o nome, endereço e telefone do cliente usando `scanf`.

4. Escreve as informações do cliente no arquivo "clientes.txt" usando `fprintf`, formatando os dados para cada linha correspondente ao código, nome, endereço e telefone do cliente.

5. Fecha o arquivo após a conclusão da escrita.

6. Exibe uma mensagem de confirmação informando que o cliente foi cadastrado com sucesso.

Exemplo de Uso:

cadastrarCliente();

Notas:

- Certifique-se de que o arquivo "clientes.txt" esteja acessível e no formato esperado para que os dados sejam corretamente armazenados.

- A função `gerarCodigo("clientes")` é utilizada para obter um código único para cada novo cliente cadastrado.

Função `cadastrarFuncionario`

Descrição:

Realiza o cadastro de um novo funcionário, coletando informações como nome, telefone, cargo e salário, e registrando esses dados em um arquivo "funcionarios.txt".

Assinatura:

void cadastrarFuncionario();

Comportamento:

1. Abre o arquivo "funcionarios.txt" no modo de adição (`"a"`). Se o arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e retorna.

2. Obtém o próximo código disponível para funcionários chamando a função `gerarCodigo("funcionarios")`.

3. Solicita ao usuário que insira o nome, telefone, cargo e salário do funcionário usando `scanf`.

4. Escreve as informações do funcionário no arquivo "funcionarios.txt" usando `fprintf`, formatando os dados para cada linha correspondente ao código, nome, telefone, cargo e salário do funcionário.

5. Fecha o arquivo após a conclusão da escrita.

6. Exibe uma mensagem de confirmação informando que o funcionário foi cadastrado com sucesso.

Exemplo de Uso:

cadastrarFuncionario();

Notas:

- Certifique-se de que o arquivo "funcionarios.txt" esteja acessível e no formato esperado para que os dados sejam corretamente armazenados.

- A função `gerarCodigo("funcionarios")` é utilizada para obter um código único para cada novo funcionário cadastrado.

Função `cadastrarQuarto`

Descrição:

Realiza o cadastro de um novo quarto, coletando informações como capacidade máxima de hóspedes, valor da diária e definindo o status inicial como "desocupado". Os dados são registrados no arquivo "quartos.txt".

Assinatura:

void cadastrarQuarto();

Comportamento:

1. Abre o arquivo "quartos.txt" no modo de adição (`"a"`). Se o arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e retorna.

2. Obtém o próximo número disponível para o quarto chamando a função `gerarCodigo("quartos")`.

3. Solicita ao usuário que insira a capacidade máxima de hóspedes e o valor da diária usando `scanf`.

4. Define o status inicial do quarto como "desocupado" usando `strcpy`.

5. Escreve as informações do quarto no arquivo "quartos.txt" usando `fprintf`, formatando os dados para cada linha correspondente ao número do quarto, capacidade máxima de hóspedes, valor da diária e status do quarto.

6. Fecha o arquivo após a conclusão da escrita.

7. Exibe uma mensagem de confirmação informando que o quarto foi cadastrado com sucesso.

Exemplo de Uso:

cadastrarQuarto();

Notas:

- Certifique-se de que o arquivo "quartos.txt" esteja acessível e no formato esperado para que os dados sejam corretamente armazenados.

- A função `gerarCodigo("quartos")` é utilizada para obter um número único para cada novo quarto cadastrado.

Função `cadastrarEstadia`

Descrição:

Realiza o cadastro de uma nova estadia, coletando informações como data de entrada, data de saída, número de diárias, código do cliente, número do quarto e número de hóspedes. Verifica a disponibilidade e capacidade do quarto, atualiza seu status para "ocupado" se estiver livre e registra os dados da estadia no arquivo "estadias.txt".

Assinatura:

void cadastrarEstadia();

Comportamento:

1. Abre o arquivo "estadias.txt" no modo de adição (`"a"`) e "quartos.txt" no modo de leitura e escrita (`"r+"`). Se algum arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e fecha os arquivos abertos antes de retornar.

2. Obtém o próximo código disponível para estadias chamando a função `gerarCodigo("estadias")`.

3. Solicita ao usuário que insira a data de entrada e a data de saída da estadia usando `scanf`.

4. Calcula o número de diárias entre a data de entrada e a data de saída usando a função `calcularDiarias`.

5. Solicita ao usuário que insira o código do cliente, número do quarto e número de hóspedes usando `scanf`.

6. Busca pelo número do quarto no arquivo "quartos.txt" para verificar se está desocupado e se possui capacidade suficiente para os hóspedes. Se encontrar um quarto adequado, atualiza seu status para "ocupado".

7. Registra as informações da estadia no arquivo "estadias.txt" usando `fprintf`, formatando os dados para cada linha correspondente ao código, datas, número de diárias, código do cliente, número do quarto e número de hóspedes.

8. Fecha os arquivos após a conclusão da escrita.

9. Exibe uma mensagem de confirmação informando que a estadia foi cadastrada com sucesso ou uma mensagem de erro se o quarto não foi encontrado, está ocupado ou não suporta o número de hóspedes informado.

Exemplo de Uso:

cadastrarEstadia();

Notas:

- Certifique-se de que os arquivos "estadias.txt" e "quartos.txt" estejam acessíveis e no formato esperado para que os dados sejam corretamente armazenados e atualizados.

- A função `gerarCodigo("estadias")` é utilizada para obter um código único para cada nova estadia cadastrada.

- A função `calcularDiarias` é utilizada para determinar o número de diárias com base nas datas de entrada e saída fornecidas.

Função `darBaixaEstadia`

Descrição:

Realiza a baixa de uma estadia existente, identificada pelo código fornecido pelo usuário. Atualiza o status do quarto associado para "desocupado" no arquivo "quartos.txt" e remove a estadia correspondente do arquivo "estadias.txt".

Assinatura:

void darBaixaEstadia();

Comportamento:

1. Abre o arquivo "estadias.txt" no modo de leitura (`"r"`), "temp_estadias.txt" no modo de escrita (`"w"`) para um arquivo temporário e "quartos.txt" no modo de leitura e escrita (`"r+"`). Se algum arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e fecha os arquivos abertos antes de retornar.

2. Solicita ao usuário que insira o código da estadia a ser baixada usando `scanf`.

3. Percorre o arquivo "estadias.txt" linha por linha, buscando pelo código da estadia fornecido. Se encontrada, atualiza o status do quarto correspondente para "desocupado" no arquivo "quartos.txt".

4. Escreve as informações não relacionadas à estadia a ser removida no arquivo temporário "temp_estadias.txt".

5. Fecha os arquivos após a conclusão da leitura e escrita.

6. Se a estadia for encontrada e baixada com sucesso, remove o arquivo "estadias.txt" e renomeia o arquivo temporário "temp_estadias.txt" para "estadias.txt". Caso contrário, remove o arquivo temporário.

7. Exibe uma mensagem de confirmação informando que a estadia foi dada baixa com sucesso ou uma mensagem de erro se a estadia não foi encontrada.

Exemplo de Uso:

darBaixaEstadia();

Notas:

- Certifique-se de que os arquivos "estadias.txt" e "quartos.txt" estejam acessíveis e no formato esperado para que os dados sejam corretamente lidos, atualizados e removidos.

- A função atualiza o status do quarto para "desocupado" apenas se a estadia correspondente for encontrada e removida com sucesso.

Função `inicializarArquivos`

Descrição:

Inicializa os arquivos de dados necessários para o sistema, criando arquivos vazios ("clientes.txt", "funcionarios.txt", "quartos.txt", "estadias.txt") se eles ainda não existirem.

Assinatura:

void inicializarArquivos();

Comportamento:

1. Abre cada um dos arquivos ("clientes.txt", "funcionarios.txt", "quartos.txt", "estadias.txt") no modo de escrita (`"w"`). Se um arquivo já existir, ele será truncado (tornando-se vazio). Se não existir, um novo arquivo vazio será criado.

2. Fecha cada arquivo após sua inicialização.

Exemplo de Uso:

inicializarArquivos();

Notas:

- Esta função é útil para garantir que os arquivos necessários para o funcionamento do sistema estejam presentes e vazios (ou criados, se não existirem) no início da execução.

- Certifique-se de que o programa tenha permissão para criar e modificar os arquivos no diretório especificado.

Função `buscarClientePorCodigo`

Descrição:

Busca e imprime os dados de um cliente com base no código fornecido, lendo as informações do arquivo "clientes.txt".

Assinatura:

void buscarClientePorCodigo(int codigo);

Parâmetros:

- `codigo`: Código do cliente a ser buscado.

Comportamento:

1. Abre o arquivo "clientes.txt" no modo de leitura (`"r"`). Se o arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e retorna.

2. Declara uma estrutura `Cliente` para armazenar os dados do cliente encontrado e uma variável `encontrado` para indicar se o cliente foi encontrado.

3. Lê cada linha do arquivo usando `fgets` e verifica se o código lido corresponde ao código fornecido pelo usuário.

4. Se um cliente com o código fornecido for encontrado:

   - Lê e armazena o nome, endereço e telefone do cliente usando `sscanf` após ler as linhas correspondentes do arquivo.

   - Imprime os dados do cliente na saída padrão.

   - Utiliza `HEADER_SEPARATOR` para separar os dados visualmente.

5. Fecha o arquivo após concluir a leitura.

6. Se o cliente não for encontrado, exibe uma mensagem informando que o cliente com o código especificado não foi encontrado.

Exemplo de Uso:

buscarClientePorCodigo(101);

Notas:

- Certifique-se de que o arquivo "clientes.txt" esteja acessível e no formato esperado para que os dados do cliente sejam corretamente lidos e impressos.

- Esta função é útil para consultas rápidas de informações de clientes com base em seus códigos.

Função `buscarFuncionarioPorCodigo`

Descrição:

Busca e imprime os dados de um funcionário com base no código fornecido, lendo as informações do arquivo "funcionarios.txt".

Assinatura:

void buscarFuncionarioPorCodigo(int codigo);

Parâmetros:

- `codigo`: Código do funcionário a ser buscado.

Comportamento:

1. Abre o arquivo "funcionarios.txt" no modo de leitura (`"r"`). Se o arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e retorna.

2. Declara uma estrutura `Funcionario` para armazenar os dados do funcionário encontrado e uma variável `encontrado` para indicar se o funcionário foi encontrado.

3. Lê cada linha do arquivo usando `fgets` e verifica se o código lido corresponde ao código fornecido pelo usuário.

4. Se um funcionário com o código fornecido for encontrado:

   - Lê e armazena o nome, telefone, cargo e salário do funcionário usando `sscanf` após ler as linhas correspondentes do arquivo.

   - Imprime os dados do funcionário na saída padrão.

   - Utiliza `HEADER_SEPARATOR` para separar os dados visualmente.

5. Fecha o arquivo após concluir a leitura.

6. Se o funcionário não for encontrado, exibe uma mensagem informando que o funcionário com o código especificado não foi encontrado.

Exemplo de Uso:

buscarFuncionarioPorCodigo(201);

Notas:

- Certifique-se de que o arquivo "funcionarios.txt" esteja acessível e no formato esperado para que os dados do funcionário sejam corretamente lidos e impressos.

- Esta função é útil para consultas rápidas de informações de funcionários com base em seus códigos.

Função `buscarEstadiasPorCliente`

Descrição:

Busca e imprime as estadias de um cliente com base no código do cliente fornecido, lendo as informações do arquivo "estadias.txt".

Assinatura:

void buscarEstadiasPorCliente(int codigoCliente);

Parâmetros:

- `codigoCliente`: Código do cliente cujas estadias estão sendo buscadas.

Comportamento:

1. Abre o arquivo "estadias.txt" no modo de leitura (`"r"`). Se o arquivo não puder ser aberto, exibe uma mensagem de erro usando `perror` e retorna.

2. Declara uma estrutura `Estadia` para armazenar os dados das estadias encontradas e uma variável `encontrado` para indicar se foram encontradas estadias para o cliente.

3. Imprime um cabeçalho indicando as estadias do cliente com o código fornecido.

4. Lê cada linha do arquivo usando `fgets` e verifica se o código de cliente da estadia lida corresponde ao código fornecido pelo usuário.

5. Se uma estadia para o cliente com o código fornecido for encontrada:

   - Lê e armazena a data de entrada, data de saída, quantidade de diárias, número do quarto e número de hóspedes da estadia usando `sscanf` após ler as linhas correspondentes do arquivo.

   - Imprime os detalhes da estadia na saída padrão.

   - Utiliza `HEADER_SEPARATOR` para separar as estadias visualmente.

6. Fecha o arquivo após concluir a leitura.

7. Se nenhuma estadia for encontrada para o cliente, exibe uma mensagem informando sobre a ausência de estadias.

Exemplo de Uso:

buscarEstadiasPorCliente(301);

Notas:

- Certifique-se de que o arquivo "estadias.txt" esteja acessível e no formato esperado para que os dados das estadias sejam corretamente lidos e impressos.

- Esta função é útil para consultar as estadias de um cliente específico com base no seu código.

Função `menu`

Descrição:

Exibe um menu interativo com opções para cadastrar clientes, funcionários, quartos, estadias, dar baixa em estadias, buscar clientes por código, buscar funcionários por código, buscar estadias por cliente ou sair do programa.

Assinatura:

void menu();

Comportamento:

1. Exibe um menu de opções numeradas de 1 a 8 e a opção 0 para sair.

2. Solicita ao usuário que escolha uma opção digitando um número.

3. Com base na opção escolhida, chama a função correspondente:

- `1`: `cadastrarCliente()`
- `2`: `cadastrarFuncionario()`
- `3`: `cadastrarQuarto()`
- `4`: `cadastrarEstadia()`
- `5`: `darBaixaEstadia()`
- `6`: `buscarClientePorCodigo()`
- `7`: `buscarFuncionarioPorCodigo()`
- `8`: `buscarEstadiasPorCliente()`
- `0`: Encerra o loop do menu e imprime "Saindo...".

4. Se o usuário escolher uma opção inválida, exibe "Opcao invalida!" e continua exibindo o menu até que a opção `0` seja escolhida.

Exemplo de Uso:

menu();

Notas:

- Este menu proporciona uma interface simples e direta para interação com as funcionalidades do sistema.

- Cada função chamada pelo menu executa operações específicas, como cadastros e consultas, baseadas na escolha do usuário.

Testes:

Casos de teste do software:

3. Casos de Teste das Funcionalidades

Função gerarCodigo

- **Teste 1: Gerar código para cliente**
  - **Entrada:** "clientes"
  - **Saída Esperada:** Próximo código disponível para cliente.
  - **Resultado:** Pass/Fail

- **Teste 2: Gerar código para funcionário**
  - **Entrada:** "funcionarios"
  - **Saída Esperada:** Próximo código disponível para funcionário.
  - **Resultado:** Pass/Fail

- **Teste 3: Gerar código para quarto**
  - **Entrada:** "quartos"
  - **Saída Esperada:** Próximo número disponível para quarto.
  - **Resultado:** Pass/Fail

- **Teste 4: Gerar código para estadia**
  - **Entrada:** "estadias"
  - **Saída Esperada:** Próximo código disponível para estadia.
  - **Resultado:** Pass/Fail

- **Teste 5: Tipo inválido**
  - **Entrada:** "invalido"
  - **Saída Esperada:** -1
  - **Resultado:** Pass/Fail

Função calcularDiarias

- **Teste 1: Datas válidas**
  - **Entrada:** "10/06/2024", "15/06/2024"
  - **Saída Esperada:** 5
  - **Resultado:** Pass/Fail

- **Teste 2: Data de saída anterior à data de entrada**
  - **Entrada:** "15/06/2024", "10/06/2024"
  - **Saída Esperada:** Valor negativo
  - **Resultado:** Pass/Fail

- **Teste 3: Datas no mesmo dia**
  - **Entrada:** "10/06/2024", "10/06/2024"

    - **Saída Esperada:** 0
    - **Resultado:** Pass/Fail

Função cadastrarCliente

- **Teste 1: Cadastro de cliente válido**
    - **Entrada:** Nome, endereço e telefone
    - **Saída Esperada:** Dados registrados corretamente em clientes.txt
    - **Resultado:** Pass/Fail

Função cadastrarFuncionario

- **Teste 1: Cadastro de funcionário válido**
    - **Entrada:** Nome, telefone, cargo e salário
    - **Saída Esperada:** Dados registrados corretamente em funcionarios.txt
    - **Resultado:** Pass/Fail

Função cadastrarQuarto

- **Teste 1: Cadastro de quarto válido**
    - **Entrada:** Capacidade máxima e valor da diária
    - **Saída Esperada:** Dados registrados corretamente em quartos.txt
    - **Resultado:** Pass/Fail

Função cadastrarEstadia

- **Teste 1: Cadastro de estadia válida**
    - **Entrada:** Data de entrada, data de saída, código do cliente, número do quarto, número de hóspedes
    - **Saída Esperada:** Dados registrados corretamente em estadias.txt e status do quarto atualizado em quartos.txt
    - **Resultado:** Pass/Fail

Função darBaixaEstadia

- **Teste 1: Baixa de estadia válida**
    - **Entrada:** Código da estadia
    - **Saída Esperada:** Estadia removida de estadias.txt e status do quarto atualizado para "desocupado" em quartos.txt
    - **Resultado:** Pass/Fail

Função buscarClientePorCodigo

- **Teste 1: Cliente existente**
    - **Entrada:** Código do cliente

    - **Saída Esperada:** Dados do cliente impressos corretamente
    - **Resultado:** Pass/Fail

- **Teste 2: Cliente inexistente**
    - **Entrada:** Código do cliente não registrado
    - **Saída Esperada:** Mensagem informando que o cliente não foi encontrado
    - **Resultado:** Pass/Fail

Função buscarFuncionarioPorCodigo

- **Teste 1: Funcionário existente**
    - **Entrada:** Código do funcionário
    - **Saída Esperada:** Dados do funcionário impressos corretamente
    - **Resultado:** Pass/Fail

- **Teste 2: Funcionário inexistente**
    - **Entrada:** Código do funcionário não registrado
    - **Saída Esperada:** Mensagem informando que o funcionário não foi encontrado
    - **Resultado:** Pass/Fail

Função buscarEstadiasPorCliente

- **Teste 1: Cliente com estadias registradas**
    - **Entrada:** Código do cliente
    - **Saída Esperada:** Dados das estadias do cliente impressos corretamente
    - **Resultado:** Pass/Fail

- **Teste 2: Cliente sem estadias registradas**
    - **Entrada:** Código do cliente sem estadias
    - **Saída Esperada:** Mensagem informando que não foram encontradas estadias para o cliente
    - **Resultado:** Pass/Fail

Função menu

- **Teste 1: Todas as opções do menu**
    - **Entrada:** Seleção de cada opção do menu
    - **Saída Esperada:** Cada função correspondente é chamada corretamente
    - **Resultado:** Pass/Fail

4. Considerações Adicionais

- **Segurança dos Dados:** É fundamental garantir a segurança dos arquivos de dados. Recomenda-se implementar verificações de permissões e backups regulares dos arquivos para evitar perda de dados.
- **Validação de Entrada:** Implementar validações robustas para os dados de entrada fornecidos pelos usuários, prevenindo erros de formatação e garantindo a integridade dos dados.

- **Interface do Usuário:** Uma interface de usuário amigável e intuitiva pode melhorar significativamente a experiência do usuário. Considerar a possibilidade de desenvolver uma interface gráfica no futuro.
- **Manutenção:** O código deve ser documentado adequadamente para facilitar a manutenção e futuras atualizações do sistema. O uso de comentários claros e concisos é essencial.
- **Escalabilidade:** O sistema foi projetado para um hotel de porte médio. Caso o número de registros cresça significativamente, pode ser necessário migrar para um sistema de banco de dados relacional para melhor performance e gestão dos dados.

5. Conclusões

O sistema de gestão hoteleira desenvolvido para o Hotel Descanso Garantido atende aos requisitos básicos de cadastro, consulta e gerenciamento de clientes, funcionários, quartos e estadias. A implementação em C proporciona um desempenho eficiente, e a documentação das funções facilita a compreensão e manutenção do código.

Com base nos testes realizados, todas as funcionalidades principais foram implementadas e testadas com sucesso. Algumas considerações adicionais foram destacadas para futuras melhorias, incluindo a segurança dos dados, validação de entrada, interface do usuário, manutenção e escalabilidade.

Recomendamos a continuidade do desenvolvimento, focando nas melhorias sugeridas e na implementação de novas funcionalidades que possam agregar valor ao sistema e aprimorar a gestão do hotel.

Código do programa e suas funções incluindo a implementação automatizada dos casos de testes.:

https://github.com/flaviojuniordev/TRABALHO-FINAL-AEDS1.git

Vídeo de apresentação : https://youtu.be/DN2VAJFmJr4