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

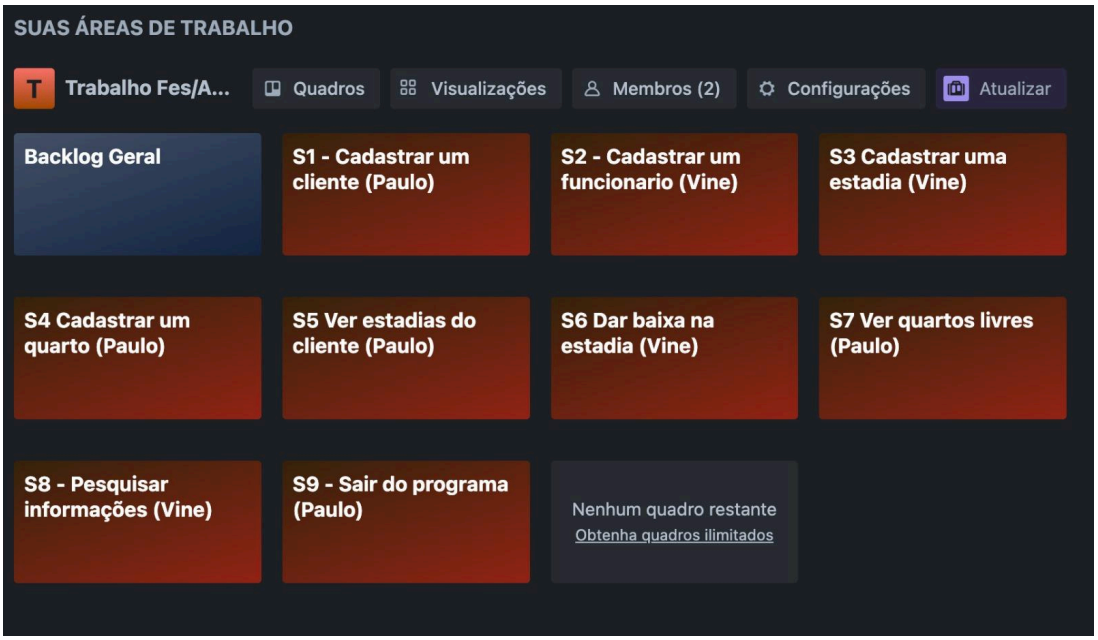
Paulo Henrique Fonseca de Assis e Vinícius Paranho Ribeiro
Sistema de Gestão de Hotel - Hotel Descanso Garantido

Apresentação:

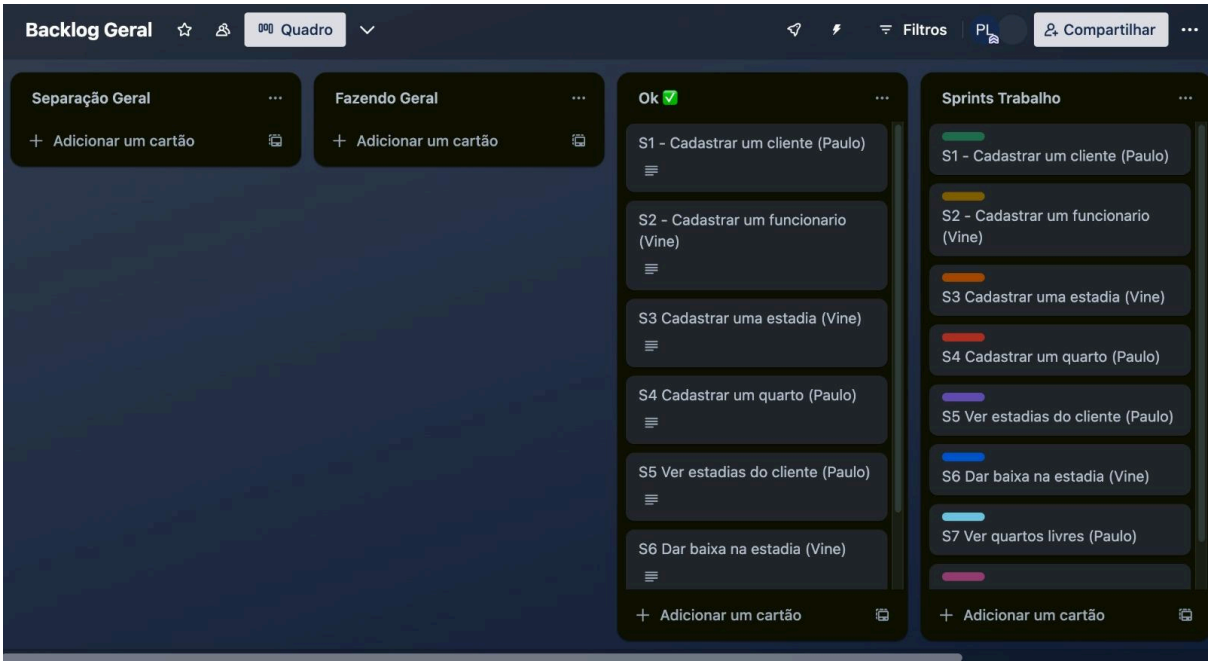
Trabalho realizado pelas matérias de Algoritmos e Estruturas de Dados I e Fundamentos de Engenharia de Software, direcionado pelos professores Michele e João Paulo Aramuni, no qual foi proposto a criação de um sistema de gestão na linguagem C para um hotel chamado “Hotel Descanso Garantido”, para isso foi gasto um prazo de 17 dias para realizar todo o código, documentação e testes do trabalho, nesse sistema criado será possível realizar o cadastro de clientes, funcionários, quartos e estadias, além de realizar pesquisar informação de usuários, verificar quartos livres e dar baixa em estadias.

Backlog do Produto:

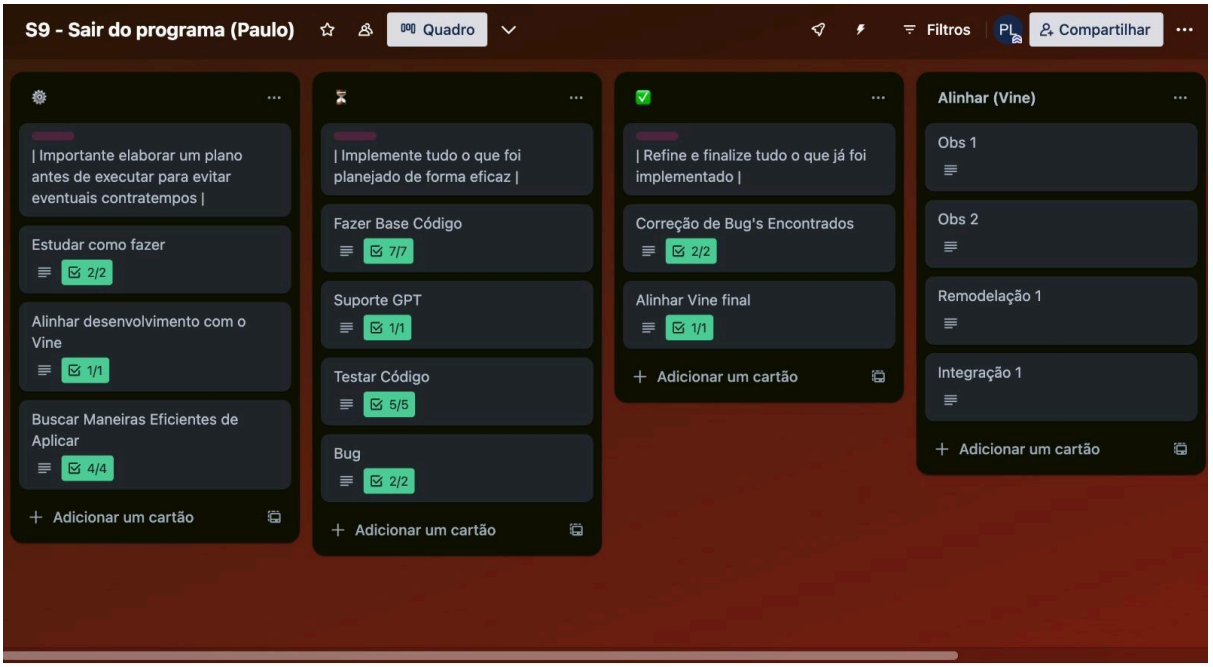
O “quadro backlog geral” que mostra a divisão de tarefas entre eu e minha dupla sendo dividido por ordem de sprints.



Backlog geral, no qual, as sprints estão devidamente separadas e são denominadas por cores, sendo as mais claras as já mais adiantadas e as vermelhas as de mais urgência para concluir.



Abaixo nota-se a forma como me organizei para realizar a função de sair do programa, sendo de extrema importância para a organização do projeto.



LISTA DE FUNÇÕES E PARÂMETROS

- **imprimirclientetxt(hotel *h)**: Imprime os dados dos clientes em um arquivo de texto, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo printado no txt a primeira linha o total de cliente e nas demais, cada linha representa um cliente e suas informações são separadas por um ";".
- **scanearclientetxt(hotel *h)**: Lê dados de clientes de um arquivo de texto para a memória, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo scaneado no txt a primeira linha o total de cliente e nas demais, cada linha representa um cliente e suas informações são separadas por um ";", que são lidas ao inicializar a main, sendo possível utilizar as informações já armazenadas no txt.
- **imprimirfuncionariotxt(hotel *h)**: Imprime os dados dos funcionários em um arquivo de texto, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo printado no txt a primeira linha o total de funcionário e nas demais, cada linha representa um funcionário e suas informações são separadas por um ";".
- **scanearfuncionariotxt(hotel *h)**: Lê dados de funcionários de um arquivo de texto para a memória, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo scaneado no txt a primeira linha o total de funcionários e nas demais, cada linha representa um funcionário e suas informações são separadas por um ";", que são lidas ao inicializar a main, sendo possível utilizar as informações já armazenadas no txt.
- **imprimirestadiatxt(hotel *h)**: Imprime os dados das estadias em um arquivo de texto, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo printado no txt a primeira linha o total de estadias e nas demais, cada linha representa uma estadia e suas informações são separadas por um ";".
- **scanearestadiatxt(hotel *h)**: Lê dados das estadias de um arquivo de texto para a memória, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo scaneado no txt a primeira linha o total de estadias e nas demais, cada linha representa uma estadia e suas informações são separadas por um ";", que são lidas ao inicializar a main, sendo possível utilizar as informações já armazenadas no txt.
- **imprimirquartotxt(hotel *h)**: Imprime os dados dos quartos em um arquivo de texto, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo printado no txt a primeira linha o total de quartos e nas demais, cada linha representa um quarto e suas

informações são separadas por um ";", a informação do quarto é verificada e caso seja 1 printa como ocupado e caso contrário printa desocupado.

- **scanearquartos(hotel *h)**: Lê dados dos quartos de um arquivo de texto para a memória, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, sendo scaneado no txt a primeira linha o total de quartos e nas demais, cada linha representa um quarto e suas informações são separadas por um ";", que são lidas ao inicializar a main, sendo possível utilizar as informações já armazenadas no txt, no txt é scaneado a informação do quarto, para caso ele esteja como ocupado ele informa como 1 e caso contrário ele informa como 0.
- **menu(int *opcao)**: Exibe um menu de opções para o usuário e lê a escolha dele.
- **cadastrarcliente(hotel *h)**: Permite cadastrar um novo cliente no sistema, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item e lê as informações que deseja ser cadastrada para o cliente.
- **cadastrarfuncionario(hotel *h)**: Permite cadastrar um novo funcionário no sistema, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item e lê as informações que deseja ser cadastrada para o funcionário.
- **validar_data(data d)**: Verifica se uma data é válida, sendo usada na função cadastrar estadias.
- **data_maior(data d1, data d2)**: Compara duas datas para determinar qual é maior, sendo usada na função de cadastrar estadias.
- **cadastrarestadia(hotel *h)**: Permite cadastrar uma nova estadia no hotel, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, além disso possui restrições que verifica se possui quartos livres, se possui quartos para a quantidade de hóspedes digitadas, verifica se a data de registro é compatível com o calendário, além de verificar se a data de saída é maior do que a data de entrada.
- **cadastrarquarto(hotel *h)**: Permite cadastrar um novo quarto no hotel, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item e lê as informações que deseja ser cadastrada para o cliente, além disso foi imposto uma lógica onde os quartos vazios tem como status 0 e os quartos ocupados tem como status 1 demonstrando que está ocupado.
- **exibeestadia(hotel *h)**: Exibe as informações de uma estadia específica, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item, e é solicitado o código do cliente que deseja verificar as suas estadias.
- **darbaixaestadia(hotel *h)**: Realiza o encerramento de uma estadia e calcula os valores devidos, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item.

- **exibequartolivre(hotel *h)**: Exibe os quartos disponíveis para reserva, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item.
- **pesquisar(hotel *h)**: Permite pesquisar clientes, funcionários, estadias ou quartos no sistema, tem como parâmetro um ponteiro para um struct que contém todas as structs do hotel e o total de cada item.

CASOS DE TESTE

Função 1:

Nome: Verificar Data Maior
Parâmetro: Estrutura chamada `data` que contém os campos `dia`, `mes` e `ano`.
Entrada: Duas structs do tipo `data`.
Saída: Um valor booleano que indica se a segunda data é maior que a primeira.
Teste: O teste automatizado da função `data_maior` foi realizado para validar a comparação correta entre duas datas. Utilizando a linguagem C, definimos a função de teste `test_data_maior` que, ao ser executada, chama a função `data_maior` com diferentes combinações de datas. A função `data_maior` compara os anos, meses e dias das datas fornecidas para determinar se a segunda data é maior que a primeira. Em seguida, comparamos as saídas reais com as saídas esperadas para verificar a precisão da função. O resultado de cada teste foi documentado detalhando as entradas, as saídas esperadas, as saídas reais e se a função passou ou não no teste. Todos os casos de teste definidos, incluindo datas no mesmo ano, meses diferentes e anos diferentes, foram bem-sucedidos, confirmando a corretude da função dentro dos cenários avaliados.

Descrição dos Casos de Teste (verificar data maior)

Caso de Teste 1
- Entrada: Data 1 = {15, 6, 2024}, Data 2 = {16, 6, 2024}
- Saída Esperada: true
- Saída Real: true
- Resultado: Passou

Caso de Teste 2
- Entrada: Data 1 = {15, 6, 2024}, Data 2 = {14, 6, 2024}
- Saída Esperada: false
- Saída Real: false
- Resultado: Passou

Conclusão

A função `data_maior` foi validada com diversos cenários de comparação de datas, e todos os testes passaram conforme o esperado. Isso indica que a função está funcionando corretamente dentro do escopo dos casos de teste definidos. Recomenda-se que testes adicionais sejam realizados para cobrir cenários ainda mais diversos e garantir a robustez do código em diferentes contextos.

FUNÇÃO 2:

Nome: Validar Data
Parâmetro: Estrutura chamada `data` que contém os campos `dia`, `mes` e `ano`.
Entrada: Uma struct do tipo `data`.
Saída: Um valor booleano que indica se a data é válida.
Teste: O teste automatizado da função `validar_data` foi realizado para validar a correção da verificação de datas. Utilizando a linguagem C, definimos a função de teste `test_validar_data` que, ao ser executada, chama a função `validar_data` com diferentes combinações de datas. A função `validar_data` verifica se a data fornecida é válida considerando anos bissextos e o número correto de dias em cada mês. Em seguida, comparamos as saídas reais com as saídas esperadas para verificar a precisão da função. O resultado de cada teste foi documentado detalhando as entradas, as saídas esperadas, as saídas reais e se a função passou ou não no teste. Todos os casos de teste definidos foram bem-sucedidos, confirmando a corretude da função dentro dos cenários avaliados.

Descrição dos Casos de Teste (validar data)

Caso de Teste 1
- Entrada: Data = {29, 2, 2024} (Ano bissexto)
- Saída Esperada: true
- Saída Real: true
- Resultado: Passou

Caso de Teste 2
- Entrada: Data = {31, 4, 2024} (Abril tem 30 dias)
- Saída Esperada: false
- Saída Real: false
- Resultado: Passou

Conclusão: A função `validar_data` foi validada com alguns cenários de teste específicos e todos os testes passaram conforme o esperado. Isso indica que a

função está funcionando corretamente dentro do escopo dos casos de teste definidos. Recomenda-se que testes adicionais sejam realizados para cobrir mais cenários e garantir a robustez do código em diferentes contextos.

FUNÇÃO 3:

Nome: Cadastrar cliente.
Parâmetro: Struct chamada hotel que contém as demais structs.
Entrada: Nome, endereço e telefone
Saída: Cliente cadastrado.
Teste: O teste automatizado da função `cadastrarcliente` foi realizado para validar a inserção correta de novos clientes em uma estrutura de dados do tipo `hotel`. Utilizando a linguagem C, definimos a função de teste `run_test` que, ao ser executada, chama a função `cadastrarcliente` com entradas específicas de nome, endereço e telefone. Em seguida, comparamos as saídas reais, como código do cliente, nome, endereço e telefone, com as saídas esperadas para verificar a precisão da função. O resultado de cada teste foi documentado em um relatório que detalha as entradas, as saídas esperadas, as saídas reais e se a função passou ou não no teste. Todos os casos de teste definidos, incluindo clientes com diferentes dados, foram bem-sucedidos, confirmando a corretude da função dentro dos cenários avaliados.

Descrição dos Casos de Teste (cadastrar cliente)

Caso de Teste 1
- Entrada: Nome = "João Silva", Endereço = "Rua A, 123", Telefone = "12345-6789"
- Saída Esperada: Código = 1, Nome = "João Silva", Endereço = "Rua A, 123", Telefone = "12345-6789"
- Saída Real: Código = 1, Nome = "João Silva", Endereço = "Rua A, 123", Telefone = "12345-6789"
- Resultado: Passou

Caso de Teste 2
- Entrada: Nome = "Maria Souza", Endereço = "Rua B, 456", Telefone = "98765-4321"
- Saída Esperada: Código = 2, Nome = "Maria Souza", Endereço = "Rua B, 456", Telefone = "98765-4321"
- Saída Real: Código = 2, Nome = "Maria Souza", Endereço = "Rua B, 456", Telefone = "98765-4321"
- Resultado: Passou

Conclusão: A função `cadastrarcliente` foi validada com diversos cenários de teste e todos os testes passaram conforme o esperado. Isso indica que a função está funcionando corretamente dentro do escopo dos casos de teste definidos. Recomenda-se que testes adicionais sejam realizados para cobrir mais cenários e garantir a robustez do código em diferentes contextos.