PONTÍFICA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

ANA LUIZA DE FREITAS RODRIGUES

JÚLIA DE SOUZA VENTURA

DOCUMENTAÇÃO DO PROJETO COMPANHIA AÉREA (VOO SEGURO)

Documentação do Trabalho Interdisciplinar das disciplinas de Algoritmos e Estruturas de Dados e Fundamentos da Engenharia de Software do curso de Engenharia de Software, Pontífica Universidade Católica de Minas Gerais, MG.

Orientadores: Ivan Luiz Vieira de Araújo

José Laerte Pires Xavier Junior

Belo Horizonte

2024

1. APRESENTAÇÃO

O sitema desenvolvido em linguagem C, tem como objetivo auxiliar no gerencimento da companhia aérea Voo Seguro, possibilitando o cadastro de passageiros, tripulação, voos e oferecer suporte no controle de reservas e baixa de assentos. Dessa forma é possível salvar toda a movimentação da companhia em arquivos.

2. BACKLOG DO PRODUTO

Para o desenvolvimento deste trabalho utilizamos a metodologia ágil Scrum para organizar o projeto em ciclos de aproximadamente 2 dias. Planejávamos cada ciclo (sprint), definindo seu backlog correspondente. Realizávamos *daily`s* (meetings) com duração média de 5 a 8 minutos para acompanhar o progresso da equipe. Ao término de cada sprint, avaliávamos o que havia sido concluído e analisávamos nosso método de trabalho para identificar oportunidades de melhoria no processo.

Na Figura 1 é possível visualizarmos o backlog geral do produto e a divisão das tarefas atribuidas a cada integrante do grupo, gerenciado a partir da ferramenta GitHub Projects para acompanhamento visual através do framework Kanban, onde separamos as tarefas nos quadros *To-do, In Progress e Done.* Visando facilitar a identificação, cada sprint foi organizada em labels distintas representadas por cores e categorias diferentes. Os itens marcados em rosa foram executas na primeira sprint, os de verde na segunda sprint, os de amarelo na terceira sprint e o de azul na quarta sprint.

Figura 1 – Time e Backlog geral do projeto

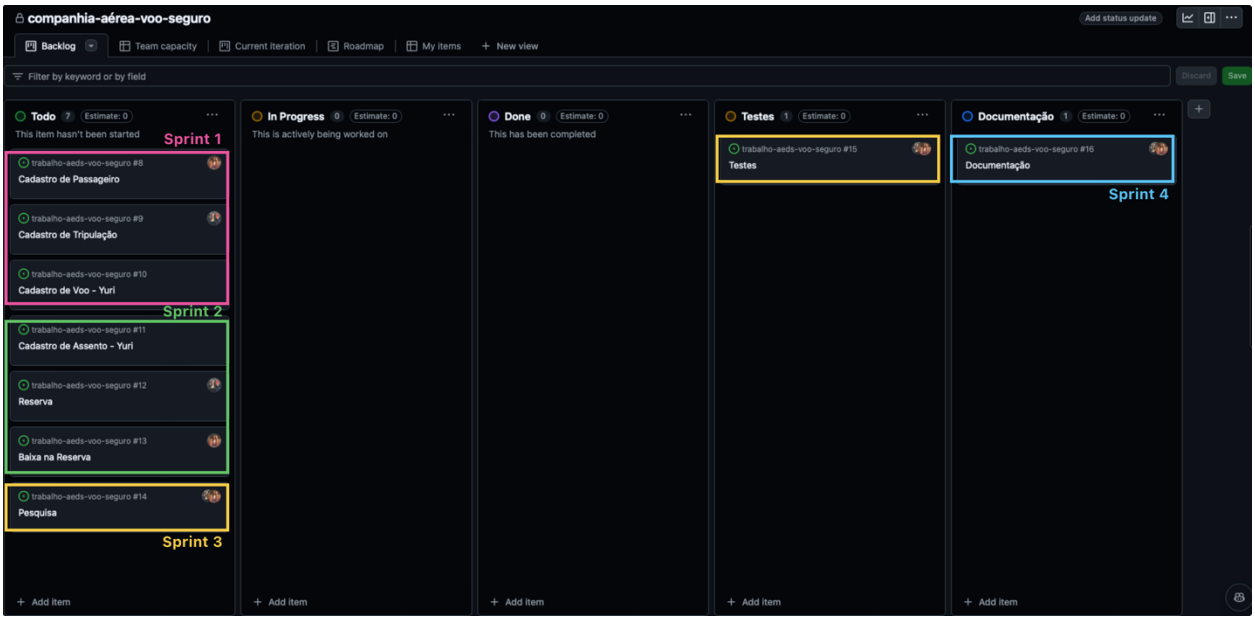


Figura 2 – Sprint 1

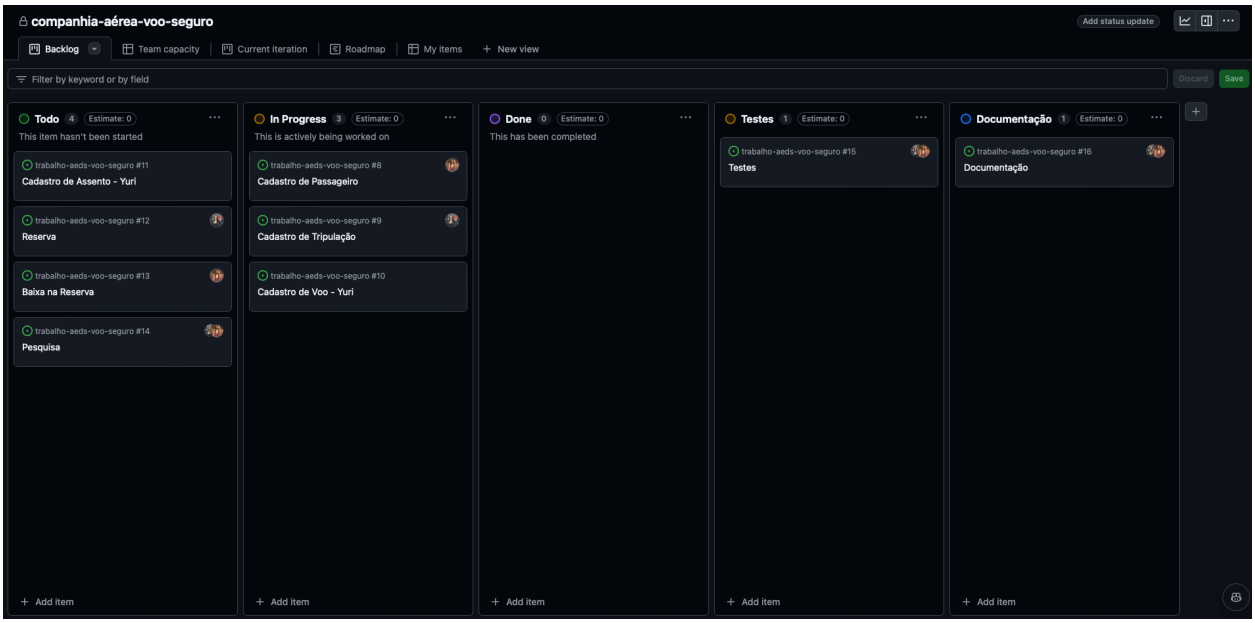


Figura 3 – Sprint 2

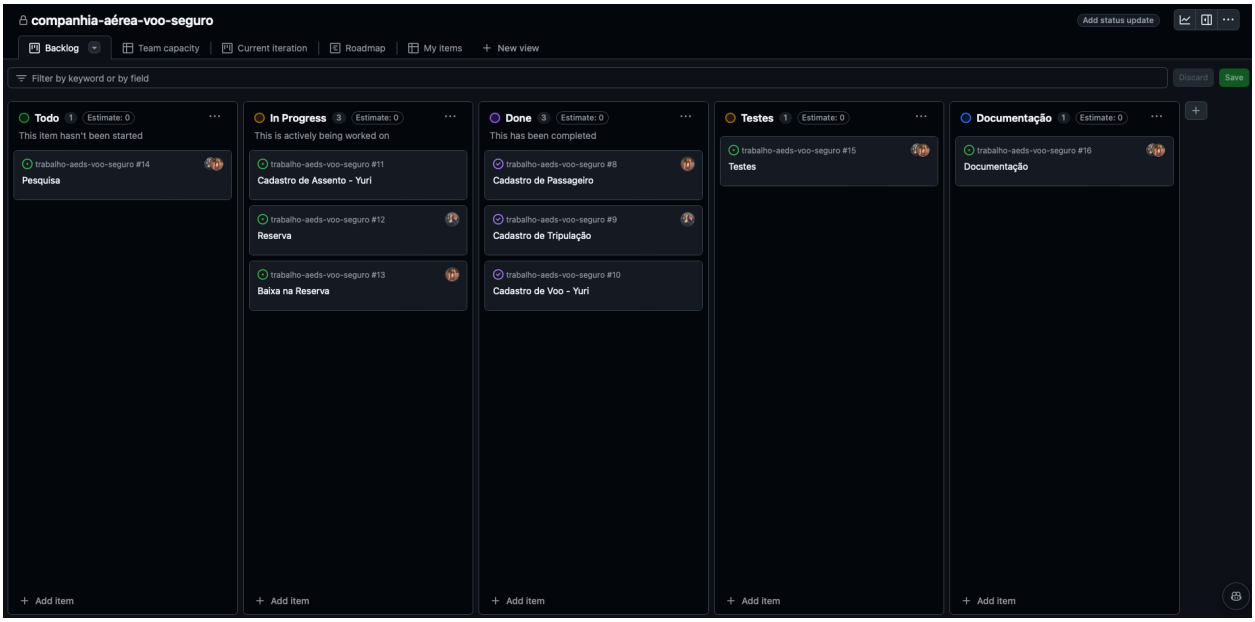


Figura 4 – Sprint 3

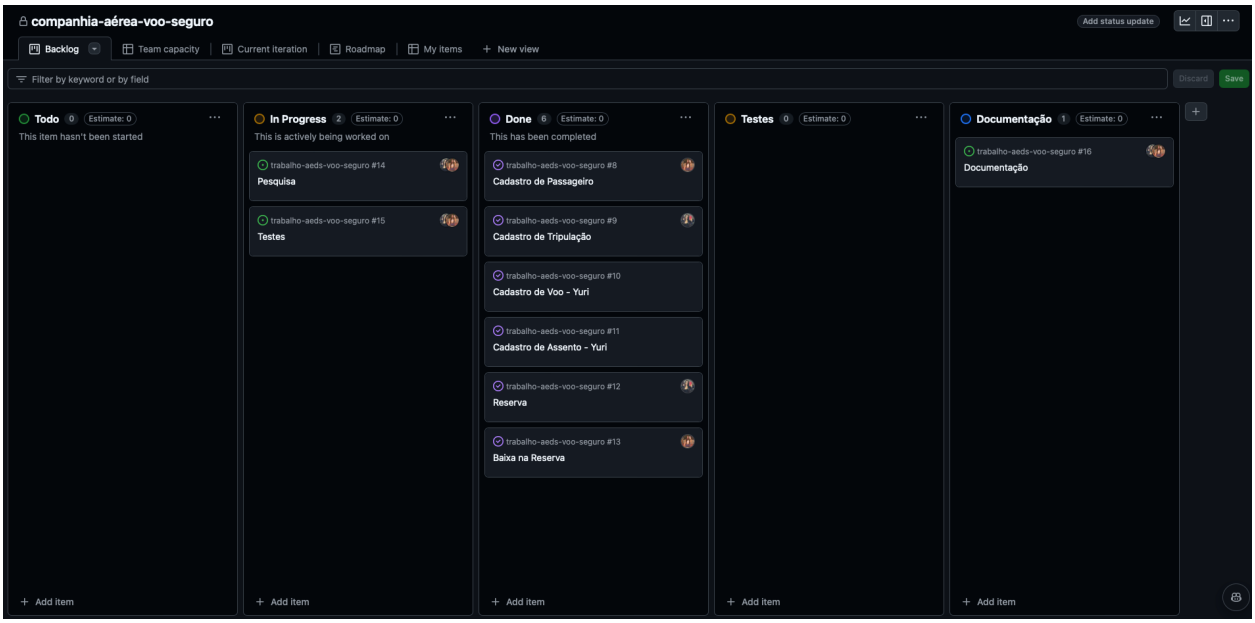


Figura 5 – Sprint 4

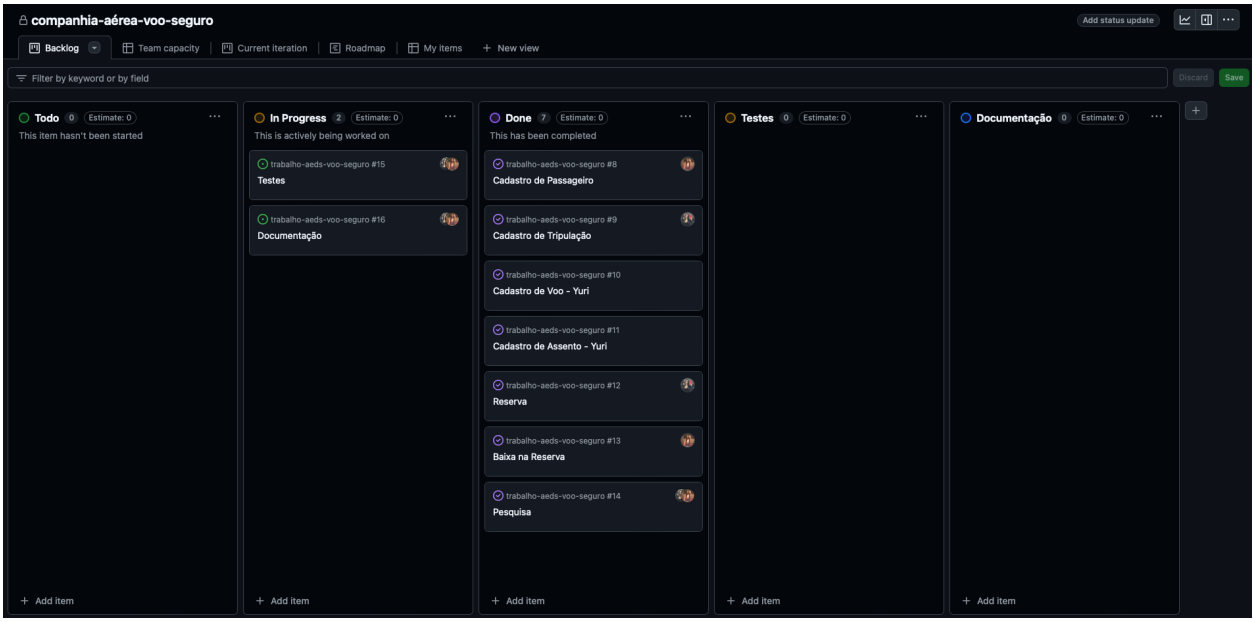
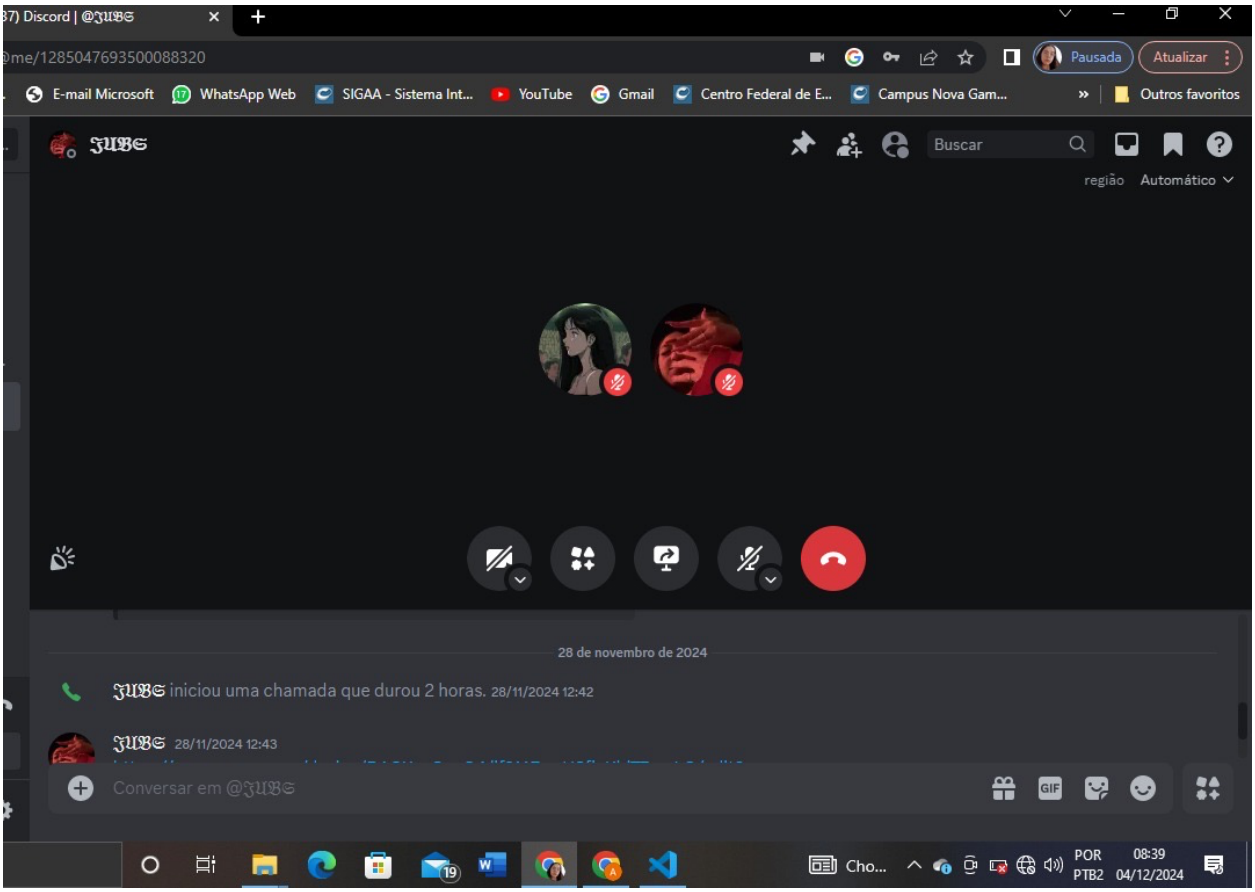


Figura 6 – Daily da Sprint 4



As Figuras 2, 3, 4 e 5 representam o status do backlog do início de execução de cada uma das sprints. A Figura 6 exemplifica uma das *daily`s* realizada durante a Sprint 4.

Datas:

- Sprint 1: 27/11 a 28/11
- Sprint 2: 29/11 a 30/11
- Sprint 3: 01/12 a 02/12
- Sprint 4: 03/12 a 04/12

3. LISTA DE ASSINATURAS DAS FUNÇÕES E PARÂMETROS

3.1. void CadastrarPassageiro(Passageiro passageiros[], int * qtdPassageiros,):

Empregada para inclusão de um novo passageiro dentro de um sistema de gerenciamento. Recebe como parâmatros um vetor 'passageiros' do tipo Passageiro e um ponteiro int que indica a quantidade de passageiros cadastrados no sistema (*qtdPassageiros). Cria uma estrutura 'novoPassageiro' do tipo Passageiro, e atribui um código identificador único a ele. Em seguida, solicita ao usuário que digite o nome, endereço e telefone do passageiro que deseja cadastrar, e armazena os dados recebidos dentro do 'novoPassageiro' recém criado. Por fim, solicita que o usuário indique se o passageiro cadastrado faz parte do Programa Fidelidade da companhia (teclando 1 caso participe, e 2 caso não participe), e armazena esta informação em 'novoPassageiro'. Após a coleta e armazenamento dos dados, armazena 'novoPassageiro' dentro do vetor passageiros na posição indicada por qtdPassageiros, e incrementa o valor desse contador. Por fim, fornece uma mensagem ao usuário indicando que o cadastro do passageiro foi concluido com sucesso e o código do passageiro cadastrado.

3.2. void cadastrarTripulacao (Tripulacao tripulantes [], int * qtdTripulantes:

Empregada para incluir um novo membro ao sistema de gerenciamento da tripulação da companhia aérea. Recebe como parâmetros um vetor tripulantes do tipo Tripulação e um ponteiro int que sinaliza o valor de tripulantes cadastrados (**qtdTripulantes**). Cria uma estrutura 'novoTripulante' do tipo Tripulante, e atribui um código identificador único a ela. Solicita que o usuário insira o nome, telefone e selecione o cargo do tripulante que deseja cadastrar(piloto, copiloto ou comissário), de acordo com as opções númericas fornecidas (1, 2 e 3). Verifica se o valor digitado

corresponde a um destes três valores numéricos: em caso negativo, gera uma mensagem ao usuário informando que o cargo digitado por ele é inválido, e a operação de cadastro não é finalizada, retornando ao menu inicial. Em caso positivo, o 'novoTripulante' é adicionado ao vetor tripulantes, o contador qtdTripulantes é incrementado e exibe-se uma mensagem ao usuário sinalizando que o cadastro foi concluido e o código do tripulante recém cadastrado.

3.3. void cadastrarVoo (Voo voos[], int * qtdVoos, Tripulacao tripulantes[], int qtdTripulantes):

Empregada para cadastro de um novo voo ao sistema de gerenciamento. Recebe como parâmetros um vetor voos do tipo Voo, um ponteiro do tipo int que indica a quantidade de voos cadastrados no sistema (qtdTripulantes), um vetor tripulantes do tipo Tripulacao e um ponteiro int que indica a quantidade de tripulantes cadastrados no sistema (qtdTripulantes). Cria uma estrutura 'novoVoo' do tipo Voo, e atribui um código identificador único a ela. Solicita que o usuário insira a data, hora, origem, destino, código do avião, tarifa do voo, o código do piloto, copiloto e do comissário e o status do voo (1 para ativo e 0 para inativo.). Testa se os códigos de piloto, copiloto e comissario fornecidos correspondem a algum código armazenado no vetor Tripulantes, e em caso negativo, fornece ao usuário uma mensagem de erro e não finaliza o cadastro do voo, retornando ao menu inicial. Em caso positivo, armazena o novoVoo no vetor voos e incrementa o contador qtdVoos, exibindo ao usuário uma mensagem de cadastro concluido com sucesso e indicando o código do novo voo cadastrado.

3.4. void cadastrarAssento(Assento assentos[], int * qtdAssentos, Voo voos[], int qtdVoos):

Empregada no cadastro de um novo assento em um voo. Recebe como parâmetros um vetor de assentos, um ponteiro para a quantidade de assentos cadastrados, um vetor de voos e a quantidade de voos cadastrados.Solicita ao usuário o código do voo e inicializa o status do assento como 0 (vazio).

Em seguida, a função verifica se o código do voo informado corresponde a algum voo presente no vetor de voos. Caso o voo não seja encontrado, é exibida uma mensagem de erro e o cadastro do assento é interrompido, com a quantidade de assentos sendo ajustada para -1, indicando que ocorreu um erro. Se o voo for encontrado, a função cria um novo assento, atribuindo-lhe o número baseado na quantidade de assentos já cadastrados. O novo assento é então adicionado ao vetor de assentos e a quantidade de assentos qtdAssentos é incrementada.

Por fim, a função exibe uma mensagem de sucesso, informando o número do novo assento cadastrado ao usuário.

3.5. void pesquisarPassageiro(Passageiro passageiros[], int qtdPassageiros):

Utilizado para buscar informações de um passageiro em uma lista cadastrada, permitindo que a busca seja feita pelo código ou pelo nome do passageiro.

Recebe como parâmetros um vetor passageiros[], do tipo Passageiro que contém os dados de todos os passageiros cadastrados e um número inteiro qtdPassageiros, que indica a quantidade total de passageiros armazenados no vetor.

O método solicita ao usuário que escolha entre buscar o passageiro pelo código ou pelo nome. Na busca por código, o sistema compara o código informado pelo usuário com os códigos existentes no vetor. Na busca por nome, o método solicita o nome completo e realiza uma comparação exata com os dados armazenados. Em ambos os casos, se o passageiro for encontrado, suas informações detalhadas são exibidas. Se não houver correspondência, o sistema informa que o passageiro não foi encontrado.

3.6. Void pesquisarTripulante(Tripulacao Tripulacao[], int qtdTripulantes):

Responsável por buscar informações de um tripulante em uma lista previamente cadastrada, permitindo que a busca seja realizada pelo código ou pelo nome do tripulante. Recebe como parametros um vetor Tripulacao [], que armazena os dados de todos os tripulantes cadastrados e um número inteiro denominado qtdTripulantes, que indica a quantidade total de tripulantes registrados no vetor. Esse parâmetro é utilizado para limitar o número de elementos percorridos durante a busca. Exibe um menu solicitando ao usuário que escolha entre buscar o tripulante pelo código ou pelo nome. Na busca por código, o sistema solicita que o usuário informe o número do código e, em seguida, verifica cada elemento do vetor até encontrar um registro correspondente. Na busca por nome, o sistema solicita o nome completo e realiza uma comparação exata com os nomes armazenados no vetor. Quando o tripulante é localizado, as informações correspondentes são exibidas; caso contrário, é exibida uma mensagem informando ao usuário que o tripulante não foi encontrado.

3.7. Void salvarVooEmArquivo(Voo voos[], int qtdVoos):

Responsável por salvar os dados de uma lista de voos em um arquivo de texto chamado voos.txt. Essa função grava as informações de cada voo no formato

estruturado, permitindo que os dados sejam armazenados permanentemente para consulta ou processamento futuro.

Recebe como parametros um vetor voos[] do tipo Voo, que contém os dados de todos os voos que devem ser salvos e um número inteiro chamado qtdVoos, que representa a quantidade de voos armazenados no vetor voos. Esse parâmetro é utilizado para controlar o número de registros que serão gravados no arquivo. A funcão testa a abertura do arquivo voos.txt no modo de apêndice. Caso o arquivo não possa ser aberto, é exibida uma mensagem de erro utilizando a função perror, e a execução é encerrada. Se o arquivo for aberto com sucesso, a função percorre o vetor voos e grava os dados de cada voo em uma nova linha no arquivo. Após o término da gravação, o arquivo é fechado para garantir a integridade dos dados.

3.8. void darBaixaReserva(Reserva reservas[], int *qtdReservas, Voo voos[], int qtdVoos, Assento assentos[], int qtdAssentos):

Responsável por cancelar uma reserva previamente registrada.

Recebe como parâmetros um vetor reservas[], que contém todas as reservas registradas, um ponteiro qtdReservas, que armazena a quantidade total de reservas registradas, um vetor voos[], que contém os dados de voos disponíveis, um inteiro qtdVoos, que indica o número total de voos no vetor, o vetor assentos[], utilizado para atualizar o status do assento reservado, e inteiro qtdAssentos, que representa a quantidade total de assentos registrados. Ele solicita que o usuário insira o código da reserva que deseja cancelar. Em seguida, a função percorre o vetor de reservas em busca de uma correspondência com o código fornecido. Caso não encontre uma reserva com o código informado, exibe uma mensagem indicando que a reserva não foi encontrada e encerra sua execução.

Se a reserva for localizada, o sistema exibe suas informações, como o código do voo, o código do passageiro e o número do assento. Após isso, o status do assento associado é alterado para disponível no vetor de assentos e no arquivo persistente de assentos. Além disso, a reserva é removida do vetor de reservas em memória e também do arquivo correspondente.

Por fim, a qtdReservas é decrementada, e uma mensagem de sucesso é exibida ao usuário.

3.9. void listarVooPassageiro(Voo voos[], int qtdVoos, Passageiro passageiros[], int qtdPassageiros):

Exibe os detalhes de todos os voos registrados no sistema para um passageiro específico, identificado pelo seu código. Recebe como parametros o vetor voos[], que contém as informações de todos os voos registrados no sistema, o inteiro qtdVoos, que representa a quantidade total de voos registrados no vetor, o vetor passageiros[], que armazena os dados de todos os passageiros cadastrados no sistema e o inteiro qtdPassageiros, que indica a quantidade total de passageiros no vetor.

Solicita ao usuário o código do passageiro e realiza uma busca no vetor de passageiros para verificar se o código fornecido corresponde a um passageiro existente. Caso o código não seja encontrado, a função exibe uma mensagem informando que o passageiro não foi localizado e encerra sua execução. Se o código do passageiro for válido, a função percorre o vetor de voos e exibe os detalhes de cada voo registrado no sistema.

3.10. void salvarDados(Passageiro passageiro[], int qtdPassageiros, Tripulacao tripulantes[], int qtdTripulantes, Voo voos[], int qtdVoos, Assento assentos[], int qtdAssentos:

Tem como objetivo salvar os dados de passageiros, tripulantes, voos e assentos em arquivos de texto, para que as informações possam ser persistidas entre execuções do programa. Ela recebe como parâmetros os vetores de dados para passageiros, tripulantes, voos e assentos, juntamente com as respectivas quantidades de elementos em cada um desses vetores.

Inicialmente, tenta abrir quatro arquivos para escrita: passageiros.txt, tripulantes.txt, voos.txt e assentos.txt. Se qualquer um desses arquivos não puder ser aberto, a função imprime uma mensagem de erro e retorna sem salvar os dados. Se os arquivos forem abertos com sucesso, a função então percorre os vetores e grava os dados no formato apropriado em cada arquivo. Para os passageiros, são salvos o código, nome, endereço, telefone e a fidelidade. Para os tripulantes, são salvos o código, nome, telefone e cargo. Para os voos, são salvos o código do voo, data, hora, origem, destino, código do avião, tarifa, e os códigos do piloto, copiloto e comissário. E para os assentos, são salvos o número do assento, o código do voo e o status do assento.

Ao final, a função fecha todos os arquivos abertos, garantindo que os dados sejam salvos corretamente.

3.11. void carregarDados(Passageiro passageiros[], int *qtdPassageiros, Tripulacao tripulantes[], int *qtdTripulantes, Voo voos[], int *qtdVoos, Assento assentos[], int *qtdAssentos):

Carrega os dados armazenados em arquivos de texto para as estruturas do programa. Ela é responsável por ler os dados dos passageiros, tripulantes, voos e assentos de arquivos de texto e armazená-los nos vetores correspondentes, além de atualizar as quantidades de passageiros, tripulantes, voos e assentos.

Tenta abrir os arquivos de texto passageiros.txt, tripulantes.txt, voos.txt e assentos.txt no modo de leitura. Se qualquer um desses arquivos não puder ser aberto, a função imprime uma mensagem de erro e retorna sem carregar os dados. Caso os arquivos sejam abertos com sucesso, a função inicializa as variáveis de quantidade (*qtdPassageiros, *qtdTripulantes, *qtdVoos, *qtdAssentos) como 0 e, em seguida, começa a ler os dados de cada arquivo usando o fscanf. Para cada arquivo, os dados são lidos linha por linha, e as informações são armazenadas nas estruturas apropriadas (passageiros, tripulantes, voos e assentos). A função aumenta a quantidade de elementos no vetor correspondente a cada vez que um novo registro é lido. Ao final da execução, os dados são carregados para os vetores de passageiros, tripulantes, voos e assentos, e as quantidades correspondentes são atualizadas. Todos os arquivos são fechados após a leitura.

4. BIBLIOTECAS UTILIZADAS

- _stdio.h_: Biblioteca padrão da linguagem c, fornece funções que executam a entrada e saída de dados no sistema, como o scanf e printf, e manipulação de dados em arquivo.
- - _stdlib.h_: Fornece funções empregadas para o controle de fluxo e término do software, manipulação de números e strings e gerenciamento de memória.
- - _string.h_: Biblioteca utilizada para a manipulação de strings (conjuntos de caracteres).
- - _assert.h_: Biblioteca que permite a verificação de condições durante a execução de um programa. Usada para a manipulação dos casos de teste implementados para o sistema, auxiliando na validação dos testes.

5. ESTRUTURAS DE DADOS

- typedef struct Passageiros: Utilizada para armazenamento dos dados dos passageiros da companhia Voo Seguro: código, nome, endereço, telefone, e se participa do Programa Fidelidade da Companhia, indicando o número de pontos.
- typedef struct Tripulação: Utilizada para o armazenamento dos dados dos tripulantes da companhia: código do tripulante, nome, telefone e cargo (piloto, copiloto ou comissário).
- typedef struct Voo: Armazena as informações dos voos da companhia: código do voo, data e hora, origem, destino, código do avião, tarifa do voo e código do piloto, copiloto e comissários do voo.
- typedef struct Assento: Armazena os dados dos assentos dos voos: código do voo em que o assento está, número do assento e status (livre ou ocupado).
- typedef struct Reserva: Armazena as informações referentes à uma reserva: número do assento reservado, código do voo em que o assento se encontra e código do passageiro que realizou a reserva.

6. TESTES

6.1. CASO DE TESTE DO SOFTWARE

Relatório dos casos de teste do software descrevendo os cenários da validação de funcionalidades e requisitos do sistema.

Cadastro de Passageiro				
Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Passageiro: Letras e números formando um nome, endereço e telefone de até 100 caracteres.	Letras e números com até 100 caracteres.	Passageiro cadastrado, o código do passageiro será gerado automaticamente e aparecerá o menu principal para a próxima etapa...	Inserir mais de 100 carateres a cada solicitação.	Aparece uma mensgem de erro e o sistema fecha automaticamente.

Cadastro de Tripulação				
Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Tripulante: Letras e números formando um nome e telefone de até 100 caracteres, número inteiro (1, 2 ou 3) de acordo com as opçoes do menu.	Letras e números com até 100 caracteres e um número inteiro sendo 1, 2 ou 3.	Tripulante cadastrado, o código do tripulante será gerado automaticamente e aparecerá o menu principal para a próxima etapa...	Inserir mais de 100 carateres a cada solicitação e inserir outro número que não seja 1, 2 ou 3.	Aparece uma mensgem de erro e o sistema fecha automaticamente caso o usuário tenha inserido mais de 100 carateres. Caso tenha inserido outro número dos apresentados, consta como cargo Inválido e é solicitado que o usuário insira as informações novamente retornando ao menu principal.

Cadastro de Voo				
Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Voo: Letras e números recebendo a data e hora do voo de até 20 caracteres, origem e destino de até 50 caracteres, código do avião, tarifa, código dos tripulantes e status do voo recebendo somente números inteiros.	Letras e números com até 50 caracteres e números inteiros, código da tripulação existente.	Voo cadastrado, o código do voo será gerado automaticamente e aparecerá o menu principal para a próxima etapa...	Inserir mais de 20 carateres na solicitação na data e hora, inserir mais de 50 caracteres na solicitação de origem e destino, e inserir o código de uma tripulação inexistente.	Aparece uma mensgem de erro e o sistema fecha automaticamente caso o usuário tenha inserido mais caracteres do que o permitido. Caso tenha inserido o código de tripulantes inexistentes, uma mensagem de tripulação não encontrada aparece solicitando que o usuário insira as informações novamente retornando ao menu principal.

Cadastro de Assento				
Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Assento: Número de um voo já cadastrado e um número informando se o assento está livre ou ocupado de acordo com o menu (0 - livre, 1 - ocupado).	Voo existente, sendo um número inteiro.	Assento cadastrado, o código do assento será gerado automaticamente e aparecerá o menu principal para a próxima etapa...	Voo inexistente.	Voo não encontrado, solicitando que o usuário insira as informações novamente retornando ao menu principal.

Fazer uma reserva				
Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Reserva: Números já cadastrados do código do voo, número do assento e código do passageiro.	Voo existente, assento existente e passageiro existente, todos sendo um número inteiro.	Reserva feita com sucesso, o código da reserva será gerado automaticamente e aparecerá o menu principal para a próxima etapa...	Voo inexistente, assento inexistente, assentos com o status ocupado e passageiro inexistente.	Passageiro não encontrado, Voo não encontrado, Assento não encontrado e assento com o status ocupado, todos solicitando que o usuário insira as informações novamente retornando ao menu principal.

Menu				
Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Menu: Números inteiros entre 0 a 9 de acordo com as opções listadas no terminal.	Números entre 0 a 9.	Execução da funcão escolhida pelo usuário.	Números menores que 0 e maiores que 9, além de valores não numéricos.	Opção inválida, solicitando que o usuário insira novamente.

6.2. RELATÓRIO DE EXECUÇÃO DE TESTES

Representação dos relatórios de testes em tabelas, onde é possível ver as entradas, classes válidas, classes inválidas e os resultados esperados.

Teste 1: Cadastro de Passageiro		
Entradas	**Classes Válidas**	**Resultado Esperado**
Opção: Char	Nomes até 100 caracteres. Endereço até 100 caracteres. Telefone até 100 caracteres.	Apresentação das informações do passageiro que foram cadastradas.

Relatório de Execução de teste		
Entradas	**Resultado**	**Aprovado**
Nome: Ana Endereço: Rua x Telefone: 31999999999	“Passageiro cadastrado com sucesso.”	Sim

Teste 2: Cadastro de Tripulação		
Entradas	**Classes Válidas**	**Resultado Esperado**
Opção: Char Int	Nomes até 100 caracteres. Telefone até 100 caracteres. Cargo do tipo inteiro de acordo com o menu.	Apresentação das informações do tripulante que foi cadastrada.

Classes Inválidas	**Resultado Esperado**
Cargos que fogem aos que foram listados.	“Cargo Inválido! Insira novamente.”

Relatório de execução de teste		
Entradas	**Resultado**	**Aprovado**
Nome: Rafael Telefone: 31888888888 Cargo: 1	“Tripulante cadastrado com sucesso.”	Sim
Nome: Carlos Telefone: 31555555555 Cargo: 5 *código interrompido*	“Cargo inválido! Insira novamente.”	Sim

Teste 3: Cadastro de Voo		
Entradas	**Classes Válidas**	**Resultado Esperado**
Opção: Char Int	Data e hora do voo. Origem e destino do voo. Código do avião. Tarifa do voo. Código do piloto, copiloto e comissário. Status do voo de acordo com o menu.	Apresentação das informações do voo que foram cadastradas.

Classes Inválidas	**Resultado Esperado**
Código da tripulação inexistente.	"Tripulação não encontrada! Insira novamente."

Relatório de execução de teste		
Entradas	**Resultado**	**Aprovado**
Data do voo: 12/12/2024 Hora do voo: 10:00 Origem: BH Destino: NY Código do avião: 456 Tarifa: 4500 Código do Piloto: 1 Código do Copiloto: 2 Código do Comissário: 3 Status: 1	"Voo cadastrado com sucesso."	Sim
Data do voo: 12/12/2024 Hora do voo: 10:00 Origem: BH Destino: NY Código do avião: 456 Tarifa: 4500 Código do Piloto: 6 Código do Copiloto: 7 Código do Comissário: 8 *código interrompido* Status: 1	"Tripulação não encontrada! Insira novamente."	Sim

Teste 4: Cadastro de Assento		
Entradas	**Classes Válidas**	**Resultado Esperado**
Opção: Int	Código do voo. Status do assento de acordo com o menu.	Apresentação das informações do assento que foram cadastradas. O número do assento é gerado automaticamente.

Classes Inválidas	**Resultado Esperado**
Código do voo inexistente.	"Voo não encontrado! Insira novamente."

Relatório de execução de teste		
Entradas	**Resultado**	**Aprovado**
Código do voo: 2 Status do assento: 0	"Assento cadastrado com sucesso."	Sim
Código do voo: 6 Status do assento: 0	"Voo não encontrado! Insira novamente."	Sim

Teste 5: Fazer Reserva		
Entradas	**Classes Válidas**	**Resultado Esperado**
Opção: Int	Código do voo. Número do assento. Código do passageiro.	Apresentação das informações da reserva que foram cadastradas.

Classes Inválidas	**Resultado Esperado**
Código do passageiro inexistente. Código do voo inexistente. Número do assento inexistente. Assento ocupado.	“Passageiro não encontrado! Insira novamente.” “Voo não encontrado! Insira novamente.” “Assento não encontrado! Insira novamente.” “Assento ocupado. Insira novamente.”

Relatório de execução de teste		
Entradas	**Resultado**	**Aprovado**
Código do passageiro: 2 Número do voo: 1 Número do assento: 5	“Reserva feita com sucesso.”	Sim
Código do passageiro: 9 *código interrompido* Número do voo: 2 Número do assento: 3	“Passageiro não encontrado! Insira novamente.”	Sim
Código do passageiro: 2 Número do voo: 9 *código interrompido* Número do assento: 6	“Voo não encontrado! Insira novamente.”	Sim
Código do passageiro: 8 Número do voo: 1 Número do assento: 1 *código interrompido*	“Assento ocupado. Insira novamente.”	Sim
Código do passageiro: 7 Número do voo: 1 Número do assento: 10 *código interrompido*	“Assento não encontrado! Insira novamente.”	Sim

Teste 6: Menu		
Entradas	**Classes Válidas**	**Resultado Esperado**
Opção: Int	Opção escolhida entre 0 a 9.	Entrada na função correspondente á opção digitada.

Classes Inválidas	**Resultado Esperado**
Opção escolhida menor que 0 ou maior que 9, e valor não numérico.	"Opção inválida! Insira novamente."

Relatório da execução		
Entradas	**Resultado**	**Aprovado**
Valor: 0	Salvar e sair	Sim
Valor: 1	Inicia a função de cadastro de passageiros	Sim
Valor: 2	Inicia a função de cadastro de tripulantes	Sim
Valor: 3	Inicia a função de cadastro de voo	Sim
Valor: 4	Inicia a função de cadastro de assento	Sim
Valor: 5	Inicia a função de pesquisa de passageiro	Sim
Valor: 6	Inicia a função de pesquisa de tripulação	Sim
Valor: 7	Inicia a função de fazer uma reserva	Sim
Valor: 8	Inicia a função de dar baixa uma reserva	Sim
Valor: 9	Inicia a função de listar os voos do passageiro	Sim

7. TESTES AUTOMATIZADOS

Como caso de teste automatizado, foi utilizado a biblioteca assert.h, permitindo a verificação de condições durante a execução de um programa. Usada para a manipulação dos casos de teste implementados para o sistema e auxiliando na validação dos testes. Funções para sua execução foram implementadas ao longo do código sendo possível escolher a opção de teste que deseja no menu principal.

7.1. void testarCadastrarPassageiro(): Simula e testa o processo de cadastro de passageiros no sistema. Ela cria um vetor para armazenar passageiros e inicializa um novo passageiro com dados fictícios, como código, nome, endereço, telefone e fidelidade. O passageiro é armazenado no vetor e o contador de passageiros

cadastrados é incrementado. Em seguida, a função valida os dados salvos usando assert, garantindo que cada campo corresponde aos valores esperados. Caso os testes sejam bem-sucedidos, uma mensagem de sucesso é exibida, juntamente com os dados do passageiro, confirmando que o cadastro foi realizado corretamente.

7.2. void testarCadastrarTripulante(): Simula e testa o funcionamento do cadastro de tripulantes no sistema. Ela cria um vetor para armazenar tripulantes e inicializa um novo tripulante com dados fictícios, como código, nome, telefone e cargo. Esse tripulante é adicionado ao vetor, e o contador de tripulantes cadastrados é incrementado. Em seguida, os dados armazenados são validados com assert, comparando cada campo com os valores esperados para garantir que o cadastro foi realizado corretamente. Se o teste for bem-sucedido, a função exibe uma mensagem de sucesso e imprime os dados do tripulante cadastrado, confirmando a integridade do processo.

7.3. void testarCadastroVoo(): Simula e testa o processo de cadastro de um voo em um sistema. Ela cria um vetor para armazenar voos e inicializa um novo voo com dados fictícios, incluindo código, data, hora, origem, destino, código do avião, tarifa, tripulantes associados e status. O voo é adicionado ao vetor, e o contador de voos cadastrados é incrementado. Em seguida, a função utiliza assert para validar se os dados armazenados no vetor correspondem aos valores esperados. Caso o teste seja bem-sucedido, uma mensagem de sucesso é exibida, acompanhada dos detalhes do voo cadastrado, garantindo que o processo de cadastro funciona corretamente.

7.4. void testarCadastrarAssento(): Simula e testa o cadastro de um assento no sistema. Ela cria um vetor para armazenar assentos e inicializa um novo assento com dados fictícios, incluindo o número do assento, o código do voo associado e o status do assento. Esse assento é adicionado ao vetor, e o contador de assentos cadastrados é incrementado. Em seguida, a função valida os dados armazenados com assert, verificando se os valores cadastrados estão corretos. Caso o teste seja bem-sucedido, uma mensagem de sucesso é exibida junto com

os detalhes do assento cadastrado, garantindo que o processo de cadastro está funcionando corretamente.

8. VIDEO

O vídeo da apresentação e funcionamento do sistema se encontra na pasta vídeo do repositório do Github com o nome *vídeo-sistema-voo.mp4*.