PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA

CURSO DE ENGENHARIA DE SOFTWARE

Disciplinas: Algoritmos e Estruturas de Dados I / Fundamentos de Engenharia de Software

Professores: Carlos Ribas e Laerte Entrega: 15/12/2024 Valor: 10 pontos

Nome dos integrantes do grupo: Matheus Guilherme Viana Pereira, Arthur Gonçalves, Pedro Henrique Maia

Introdução

O sistema **VooSeguro** foi desenvolvido para atender às necessidades de gerenciamento da companhia aérea fictícia Voo Seguro. Este software tem como objetivo centralizar e automatizar os processos de administração de voos, passageiros, tripulação e reservas, eliminando os problemas associados ao controle manual, como duplicidade de reservas e falta de organização nas informações.

O **VooSeguro** inclui funcionalidades essenciais, como:

- Cadastro de passageiros, tripulação, voos e assentos.
- Gestão de reservas com validações para garantir a integridade dos dados e evitar inconsistências.
- Controle de disponibilidade de voos e assentos em tempo real.
- Implementação de um programa de fidelidade que acumula pontos automaticamente para passageiros cadastrados.

O sistema foi desenvolvido utilizando a linguagem de programação C e segue boas práticas de Engenharia de Software, com foco em modularidade, validações robustas e documentação detalhada. Sua arquitetura permite fácil manutenção e evolução, além de proporcionar uma interface simples para os usuários finais.

Com o **VooSeguro**, busca-se otimizar a operação da companhia aérea, reduzindo erros manuais e garantindo um gerenciamento eficiente e seguro.

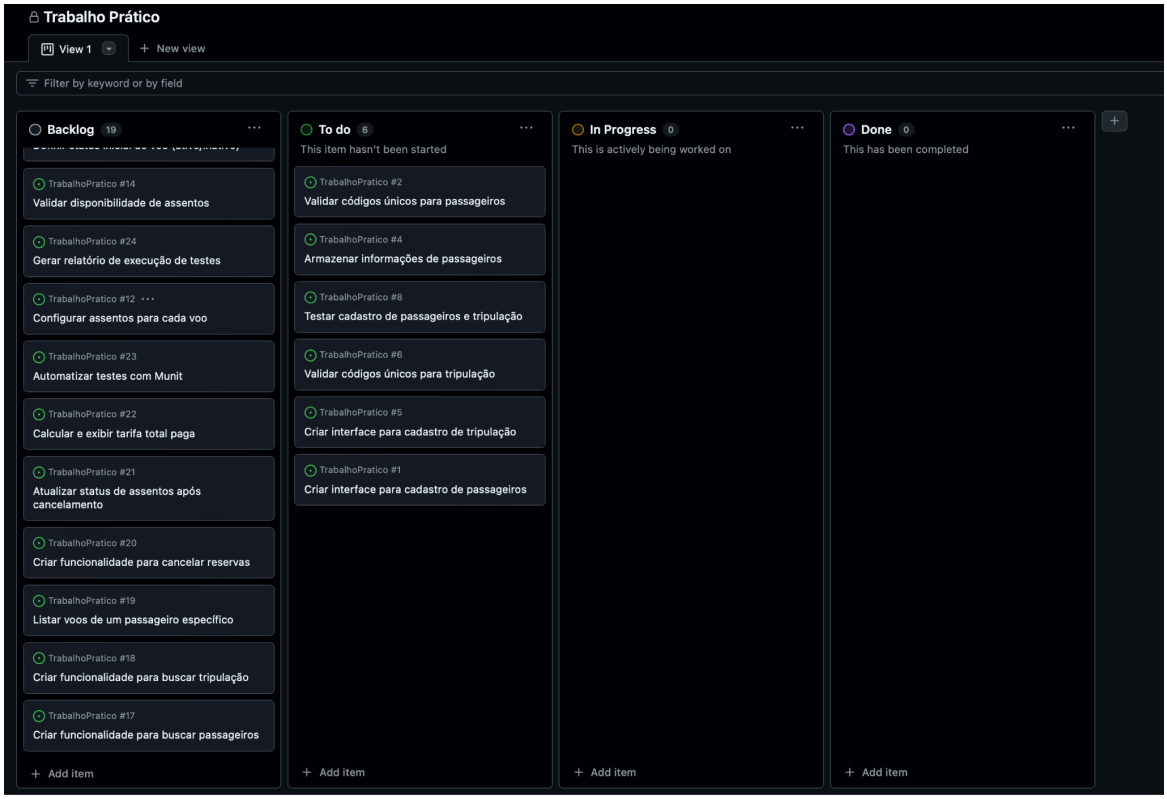
Backlog do Produto

Sprint 1: Configuração Inicial e Cadastros Básicos

Período: 3 a 4 dias
Objetivo: Estabelecer a base do sistema com funcionalidades iniciais de cadastro de passageiros e tripulação.

Quadro Inicial do Backlog

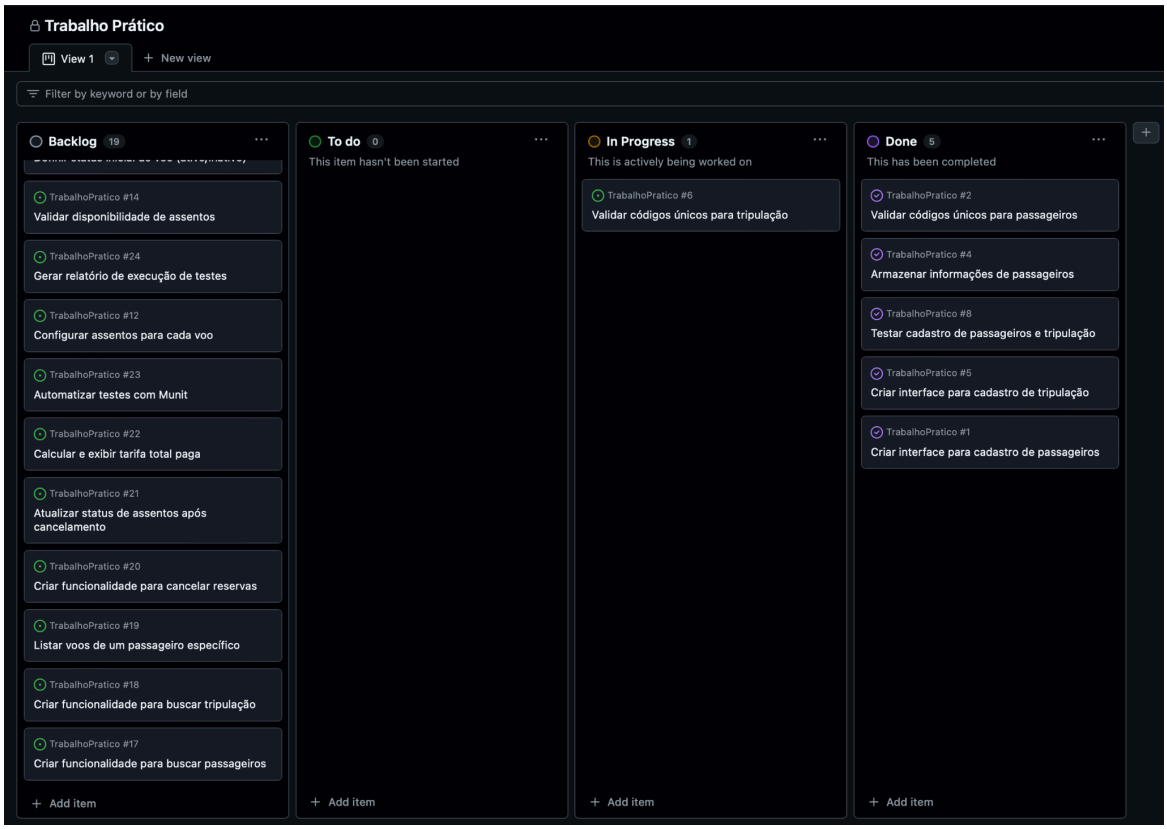
Descrição: O estado inicial do backlog exibe todas as tarefas planejadas para a Sprint 1 na coluna *To Do*. Nenhuma tarefa foi iniciada ainda.



Legenda: Estado inicial do backlog para a Sprint 1, mostrando todas as tarefas planejadas aguardando início.

Quadro de Evolução do Backlog

Descrição: Durante a Sprint 1, as tarefas foram movidas entre as colunas *To Do*, *In Progress* e *Done*. O status do quadro atualizado é apresentado abaixo.



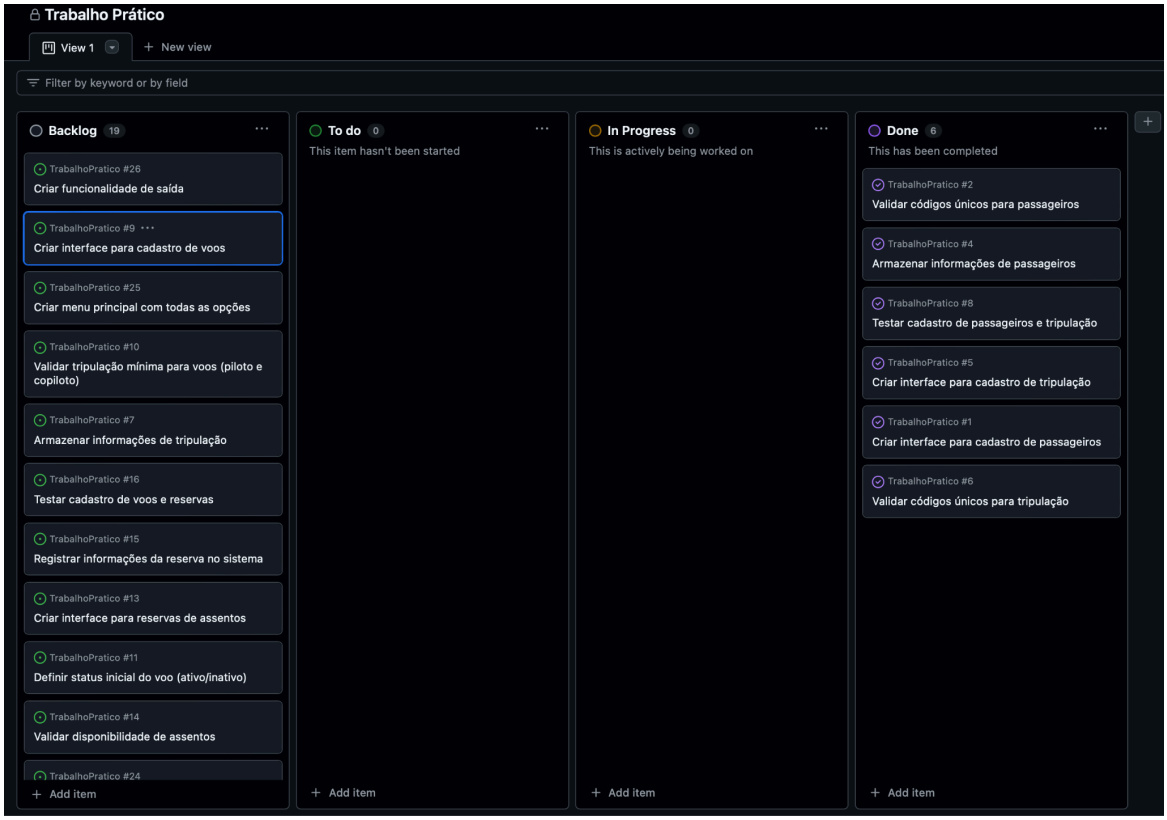
Legenda: Evolução do backlog ao final da Sprint 1, evidenciando as tarefas concluídas e em andamento.

Tarefas Realizadas na Sprint 1

Tarefa	Responsável	Status	Comentário
Criar interface para cadastro de passageiros	Pedro Henrique	Concluído	Interface implementada com os campos Nome, Endereço, Telefone e Fidelidade.
Validar códigos únicos para passageiros	Matheus Guilherme	Concluído	Validação implementada para evitar códigos duplicados.
Criar interface para cadastro de tripulação	Arthur Gonçalves	Concluído	Interface com campos Nome, Telefone e Cargo (Piloto, Copiloto, Comissário).
Testar cadastros de passageiros e tripulação	Arthur Gonçalves	Concluído	Casos de teste validados com entradas válidas e inválidas.
Armazenar informações de passageiros	Pedro Henrique	Concluído	Estrutura de dados implementada para armazenamento de passageiros.

Progresso da Sprint 1

Descrição: Apresentação gráfica do progresso da Sprint 1.



Legenda: Tarefas concluídas, em andamento e pendentes ao final da Sprint 1.

Resultados Finais da Sprint 1

Descrição: Resultado das tarefas concluídas ao final da Sprint 1, documentando as funcionalidades implementadas.

1. **Interface para cadastro de passageiros**
    - Tela funcional para o cadastro de passageiros, com campos implementados para Nome, Endereço, Telefone e Fidelidade.
2. **Interface para cadastro de tripulação**
    - Funcionalidade concluída com os campos Nome, Telefone e Cargo (Piloto, Copiloto, Comissário).
3. **Validação de códigos únicos para passageiros**
    - Validação implementada para evitar duplicidade ao cadastrar passageiros.
4. **Teste de cadastros de passageiros e tripulação**
    - Casos de teste realizados com entradas válidas e inválidas, validando o comportamento esperado.
5. **Estrutura de dados para armazenamento de passageiros**

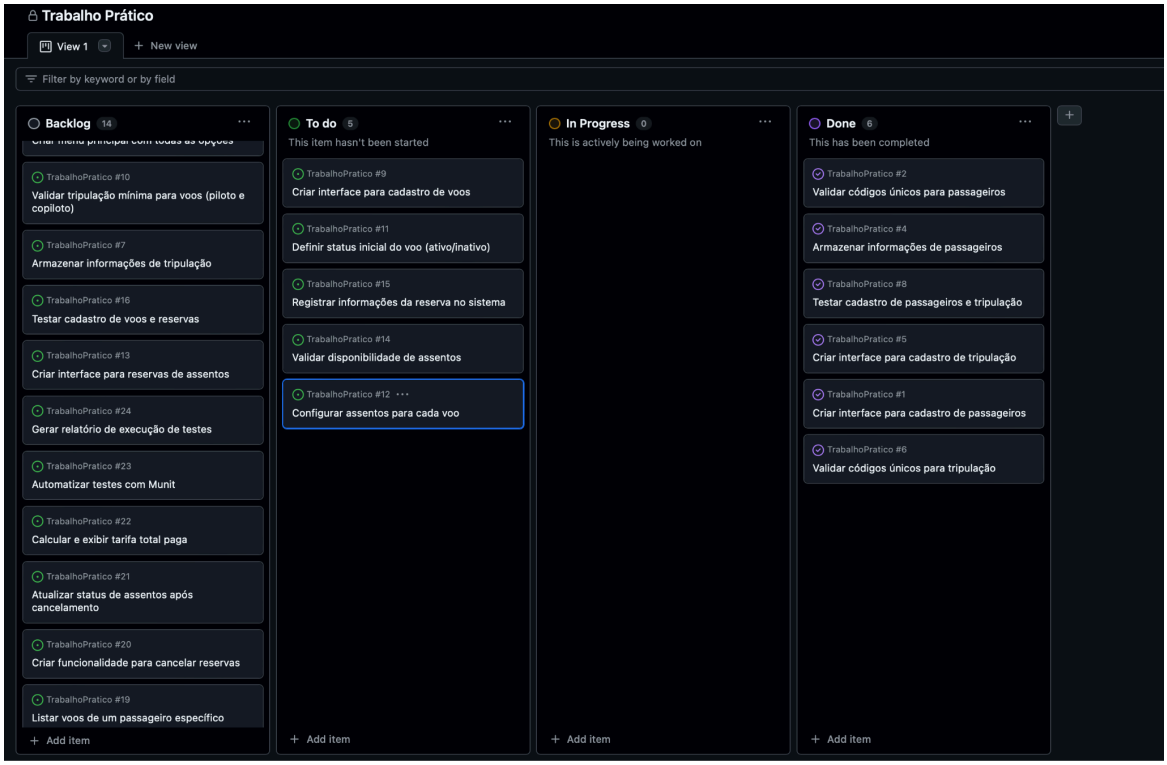
- Implementação de estrutura de dados para armazenar informações dos passageiros cadastrados.

Sprint 2: Funcionalidades de Voos e Reservas

Período: 4 a 5 dias
Objetivo: Implementar e validar as funcionalidades relacionadas ao cadastro de voos, reservas e a gestão de assentos.

Quadro Inicial do Backlog

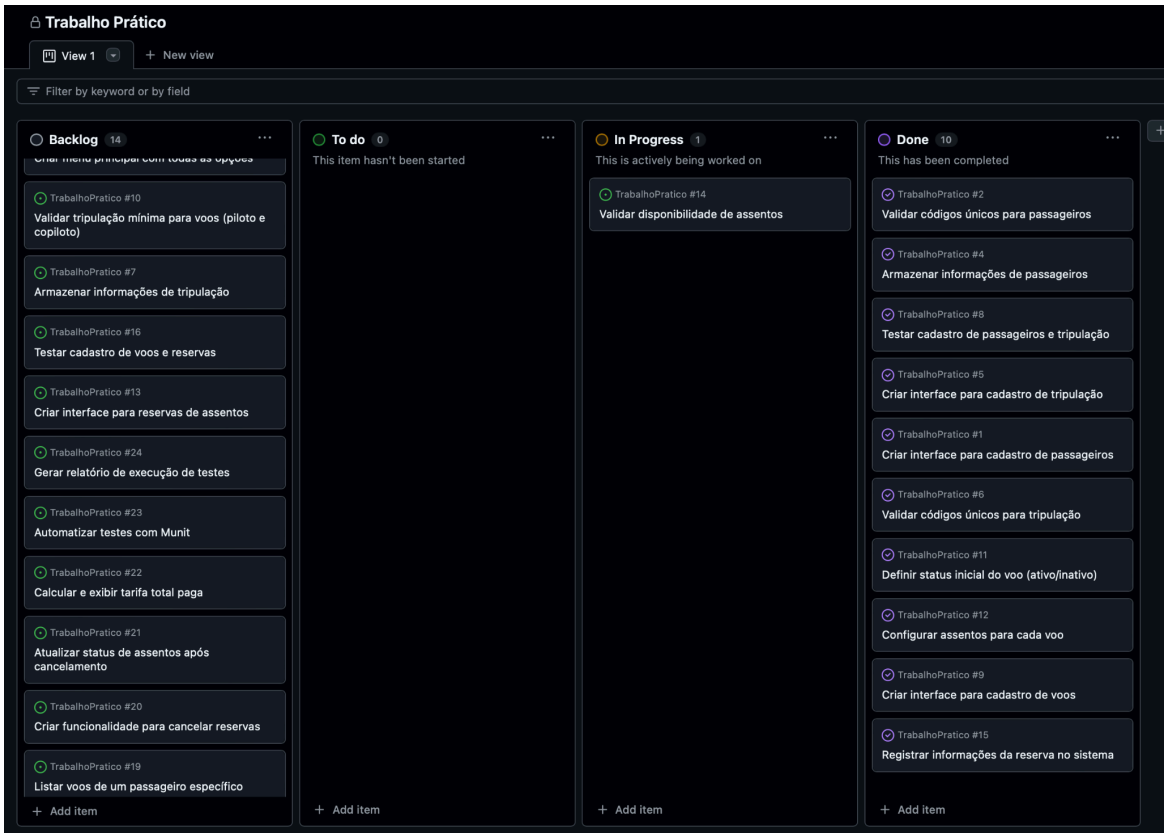
Descrição: O estado inicial do backlog exibe todas as tarefas planejadas para a Sprint 2 na coluna *To Do*. Nenhuma tarefa foi iniciada ainda.



Legenda: Estado inicial do backlog para a Sprint 2, mostrando todas as tarefas planejadas aguardando início.

Quadro de Evolução do Backlog

Descrição: Durante a Sprint 2, as tarefas foram movidas entre as colunas *To Do*, *In Progress* e *Done*. O status do quadro atualizado é apresentado abaixo.



Legenda: Evolução do backlog ao final da Sprint 2, evidenciando as tarefas concluídas e em andamento.

Tarefas Realizadas na Sprint 2

Tarefa	Responsável	Status	Comentário
Criar interface para cadastro de voos	Matheus Guilherme	Concluído	Tela funcional implementada com campos para Data, Origem, Destino e Tarifa.
Configurar status inicial do voo (ativo/inativo)	Pedro Henrique	Concluído	Validação implementada para garantir a presença de um piloto e copiloto.
Criar interface para reservas de assentos	Arthur Gonçalves	Concluído	Interface concluída, permitindo reservas por código de voo e assento.
Registrar informações da reserva no sistema	Pedro Henrique	Concluído	Reserva registrada com validação de assentos disponíveis e voos ativos.
Validar disponibilidade de assentos	Matheus Guilherme	Concluído	Garantido que os assentos ocupados não possam ser reservados novamente.

Resultados Finais da Sprint 2

Descrição: Resultado das tarefas concluídas ao final da Sprint 2, documentando as funcionalidades implementadas.

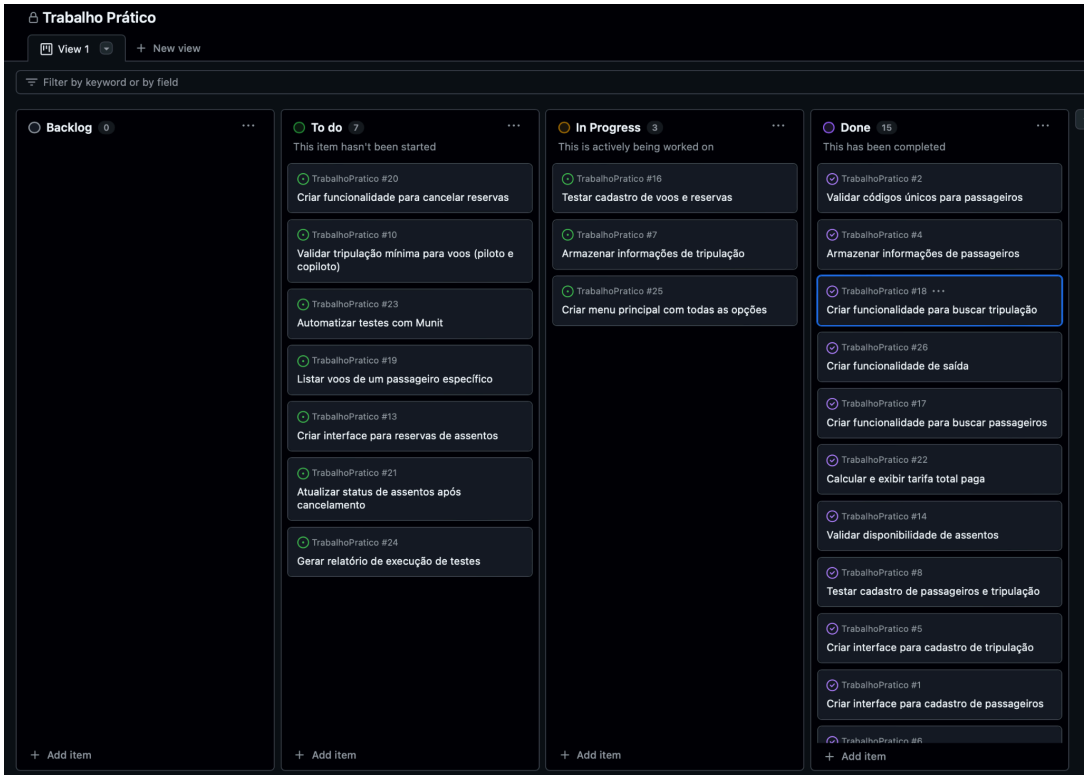
1. **Interface para cadastro de voos**
    - Tela funcional para o cadastro de voos, com campos para Data, Origem, Destino e Tarifa.
2. **Configuração do status inicial do voo**
    - Implementação concluída para validar se um voo pode ser ativado ou permanece inativo.
3. **Interface para reservas de assentos**
    - Tela funcional para registrar reservas de assentos com campos para código do voo, número do assento e código do passageiro.
4. **Validação de assentos disponíveis**
    - Implementação concluída garantindo que os assentos ocupados não possam ser reservados novamente.
5. **Registro das reservas no sistema**
    - Reserva armazenada no sistema com validações completas para voos ativos e assentos livres.

Sprint 3: Funcionalidades Avançadas e Relatórios

Período: 4 a 5 dias
Objetivo: Implementar funcionalidades avançadas, como cancelamento de reservas, geração de relatórios e automatização de testes.

Quadro Inicial do Backlog

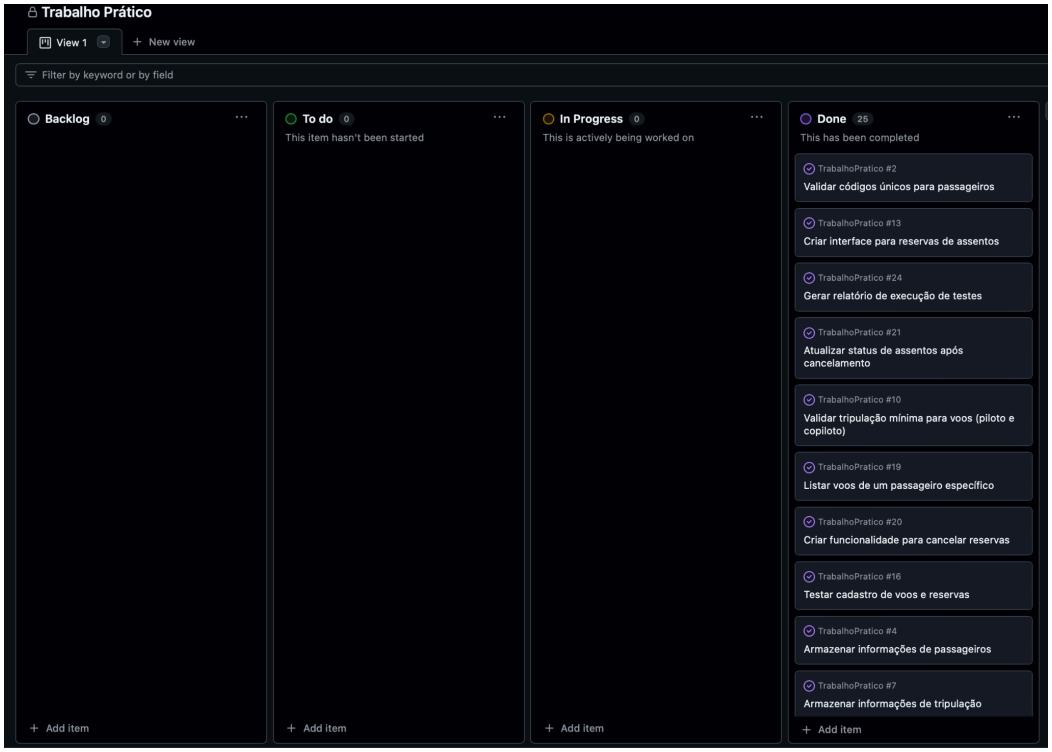
Descrição: O estado inicial do backlog da Sprint 3 exibe todas as tarefas planejadas na coluna *To Do*. Nenhuma tarefa foi iniciada ainda.



Legenda: Estado inicial do backlog para a Sprint 3, mostrando todas as tarefas planejadas aguardando início.

Quadro de Evolução do Backlog

Descrição: Durante a Sprint 3, as tarefas foram movidas entre as colunas *To Do*, *In Progress* e *Done*. O status do quadro atualizado é apresentado abaixo.



Legenda: Evolução do backlog ao final da Sprint 3, evidenciando as tarefas concluídas e em andamento.

Tarefas Realizadas na Sprint 3

Tarefa	Responsável	Status	Comentário
Criar funcionalidade para cancelar reservas	Pedro Henrique	Concluído	Permite cancelar reservas e atualizar o status do assento para 'livre'.
Atualizar status dos assentos após cancelamento	Matheus Guilherme	Concluído	Implementação para garantir que os assentos sejam corretamente atualizados.
Gerar relatório de execução de testes	Arthur Gonçalves	Concluído	Relatório gerado com resultados dos testes realizados nas funcionalidades.
Automatizar testes com MUnit	Arthur Gonçalves	Concluído	Testes automatizados para validar as funcionalidades implementadas.
Listar voos de um passageiro específico	Matheus Guilherme	Concluído	Permite consultar todos os voos de um passageiro cadastrado.

Resultados Finais da Sprint 3

Descrição: Resultado das tarefas concluídas ao final da Sprint 3, documentando as funcionalidades implementadas.

1. **Cancelar reservas e atualizar status dos assentos**

    - Funcionalidade implementada para permitir o cancelamento de reservas e garantir que os assentos sejam liberados corretamente.
2. **Relatório de execução de testes**
    - Relatório detalhado dos testes realizados, com os resultados das validações de cada funcionalidade.
3. **Automatização de testes com MUnit**
    - Testes automatizados implementados para verificar a consistência do sistema após cada atualização.
4. **Listagem de voos de um passageiro específico**
    - Funcionalidade para consultar todos os voos de um passageiro, exibindo informações detalhadas.

3. Lista de Assinaturas das Funções e Parâmetros - Sistema VooSeguro

1. Módulo de Cadastro de Passageiros (cadastrarPassageiro.h/c)

int validarNumeros(const char *str)

Propósito: Verifica se uma string contém apenas números

- Entrada: str - String a ser validada

- Retorno: 1 se contém apenas números, 0 caso contrário

- Restrições: String não pode ser nula ou vazia

int validarNome(const char *nome)

Propósito: Valida o nome do passageiro

- Entrada: nome - String com o nome a ser validado

- Retorno: 1 se válido, 0 se inválido

- Restrições: Entre 3 e 50 caracteres, apenas letras e espaços

int validarEndereco(const char *endereco)

Propósito: Valida o endereço do passageiro

- Entrada: endereco - String com o endereço a ser validado

- Retorno: 1 se válido, 0 se inválido

- Restrições: Entre 5 e 100 caracteres, deve conter números e letras

int validarTelefone(const char *telefone)

Propósito: Valida o número de telefone

- **Entrada:** telefone - String com o número a ser validado

- **Retorno:** 1 se válido, 0 se inválido

- **Restrições:** Exatamente 11 dígitos numéricos

void limparBuffer()

Propósito: Limpa o buffer do teclado

- **Funcionamento:** Remove caracteres residuais do stdin

- **Uso:** Após operações de leitura para evitar problemas de buffer

void cadastrarPassageiro()

Propósito: Realiza o cadastro completo de um novo passageiro

- Funcionamento:
  - Solicita dados do passageiro
  - Valida todas as informações
  - Armazena na estrutura global
- **Restrições:** Máximo de 100 passageiros

void listarPassageiros()

Propósito: Exibe todos os passageiros cadastrados

- Funcionamento:
  - Lista todos os dados
  - Exibe status de fidelidade
  - Mostra pontos acumulados

2. Módulo de Cadastro de Voos (cadastrarVoo.h/c)

int validarData(const char *data)

Propósito: Valida a data do voo

- Entrada: data - String no formato DD/MM/AAAA
- Retorno: 1 se válida, 0 se inválida
- Restrições:
  - Data deve ser futura
  - Formato DD/MM/AAAA
  - Valores válidos para dia e mês

int validarHora(const char *hora)

Propósito: Valida o horário do voo

- Entrada: hora - String no formato HH:MM
- Retorno: 1 se válida, 0 se inválida
- Restrições:
  - Formato 24h
  - Valores válidos (00:00 a 23:59)

int validarLocalizacao(const char *local)

Propósito: Valida cidade de origem/destino

- Entrada: local - Nome da cidade

- Retorno:** 1 se válida, 0 se inválida

- Restrições:

  - Entre 2 e 50 caracteres

  - Apenas letras e espaços

int validarTarifa(float tarifa)

Propósito: Valida o valor da tarifa

- Entrada: tarifa - Valor em reais

- Retorno: 1 se válida, 0 se inválida

- Restrições: Valor maior que zero

void limparBufferVoo()

Propósito: Limpa o buffer do teclado no módulo de voos

- Funcionamento: Remove caracteres residuais do stdin

- Uso: Após operações de leitura para evitar problemas de buffer

void cadastrarVoo()

Propósito: Realiza o cadastro de um novo voo

- Funcionamento:

  - Coleta e valida dados do voo

  - Verifica tripulação mínima

  - Configura assentos

  - Define status inicial

- Restrições: Máximo de 100 voos

void configurarAssentos(int indiceVoo)

Propósito: Inicializa os assentos de um voo

- Entrada: indiceVoo - Índice do voo na lista

- Funcionamento:

  - Marca todos os assentos como livres

  - Associa assentos ao código do voo

void listarVoos()

Propósito: Exibe todos os voos cadastrados

- Funcionamento:

  - Lista informações de cada voo

  - Mostra data, hora, origem, destino

  - Exibe status e tarifa atual

int verificarStatusVoo(int codigoVoo)

Propósito: Verifica se um voo está ativo

- Entrada: codigoVoo - Código do voo

- Retorno: 1 se ativo, 0 se inativo

- Funcionamento: Consulta status atual do voo

3. Módulo de Cadastro de Tripulação (cadastrarTripulacao.h/c)

int validarCargoTripulacao(const char *cargo)

Propósito: Valida o cargo do tripulante

- Entrada: cargo - String com o cargo

- Retorno: 1 se válido, 0 se inválido

- Restrições: Deve ser piloto, copiloto ou comissário

iint validarNomeTripulacao(const char *nome)

Propósito: Valida o nome do tripulante

- Entrada: nome - String com o nome

- Retorno: 1 se válido, 0 se inválido

- Restrições: Entre 3 e 50 caracteres, apenas letras e espaços

int validarTelefoneTripulacao(const char *telefone)

Propósito: Valida o telefone do tripulante

- **Entrada:** telefone - String com o número

- **Retorno:** 1 se válido, 0 se inválido

- **Restrições:** 10 ou 11 dígitos numéricos

void limparBufferTripulacao()

Propósito: Limpa o buffer do teclado no módulo de tripulação

- Funcionamento: Remove caracteres residuais do stdin

- Uso: Após operações de leitura para evitar problemas de buffer

void cadastrarTripulacao()

Propósito: Realiza o cadastro de um novo tripulante

- Funcionamento:
  - Coleta e valida dados
  - Verifica cargo
  - Armazena na lista
- **Restrições:** Máximo de 50 tripulantes

void listarTripulacao()

Propósito: Exibe todos os tripulantes cadastrados

- **Funcionamento:**
  - Lista dados de cada tripulante
  - Mostra cargo e contato
  - Exibe status atual

int verificarTripulacaoMinima(int codigoPiloto, int codigoCopiloto)

Propósito: Verifica tripulação mínima necessária

- Entrada:
  - codigoPiloto - Código do piloto
  - codigoCopiloto - Código do copiloto
- Retorno: 1 se atende requisitos, 0 se não atende
- Restrições: Necessário piloto e copiloto válidos

4. Módulo de Reservas (cadastrarReserva.h/c)

void limparBufferReserva()

Propósito: Limpa o buffer do teclado no módulo de reservas

- Funcionamento: Remove caracteres residuais do stdin

- Uso: Após operações de leitura para evitar problemas de buffer

int verificarDisponibilidadeAssento(int codigoVoo, int numeroAssento)

Propósito: Verifica disponibilidade de um assento

- Entrada:

  - codigoVoo - Código do voo

  - numeroAssento - Número do assento

- Retorno: 1 se disponível, 0 se ocupado

- Restrições: Assento deve existir no voo

int validarPassageiroExiste(int codigoPassageiro)

Propósito: Verifica existência do passageiro

- Entrada: codigoPassageiro - Código do passageiro

- Retorno: 1 se existe, 0 se não existe

- Funcionamento: Busca na lista de passageiros

int validarVooAtivo(int codigoVoo)

Propósito: Verifica se um voo está apto para reservas

- Entrada: codigoVoo - Código do voo

- Retorno* 1 se ativo, 0 se inativo

- Restrições: Voo deve existir e estar operacional

void cadastrarReserva()

Propósito: Realiza uma nova reserva

- Funcionamento:
  - Verifica disponibilidade
  - Valida passageiro e voo
  - Calcula valor
  - Atualiza pontos
- Restrições: Voo deve estar ativo

void baixarReserva()

Propósito: Cancela uma reserva existente

- Funcionamento:
  - Libera assento
  - Calcula reembolso
  - Atualiza sistema
- Restrições: Reserva deve existir

void listarReservasPassageiro(int codigoPassageiro)

Propósito: Lista todas as reservas de um passageiro

- Entrada: codigoPassageiro - Código do passageiro
- Funcionamento:
  - Lista todas as reservas ativas
  - Mostra detalhes dos voos

- Exibe valores e status

5. Módulo de Pesquisa (pesquisa.h/c)

void limparBufferPesquisa()

Propósito: Limpa o buffer do teclado no módulo de pesquisa

- Funcionamento: Remove caracteres residuais do stdin

- Uso: Após operações de leitura para evitar problemas de buffer

void pesquisarPassageiroPorCodigo(int codigo)

Propósito: Busca passageiro por código

- Entrada: codigo - Código do passageiro

- Funcionamento:

  - Busca passageiro específico

  - Exibe dados se encontrado

  - Mostra status de fidelidade

void pesquisarPassageiroPorNome(const char *nome)

Propósito: Busca passageiros por nome

- Entrada: nome - Nome a pesquisar

- Funcionamento:

  - Busca nomes similares

  - Exibe todos os resultados encontrados

  - Mostra dados completos

void pesquisarTripulacaoPorCodigo(int codigo)

Propósito: Busca tripulante por código

- Entrada: codigo - Código do tripulante

- Funcionamento:

  - Busca tripulante específico

  - Exibe dados se encontrado

  - Mostra cargo atual

void pesquisarTripulacaoPorNome(const char *nome)

Propósito: Busca tripulantes por nome

- Entrada: nome - Nome a pesquisar

- Funcionamento:

  - Busca nomes similares

  - Exibe todos os resultados

  - Mostra informações de cargo

void listarVoosPassageiro(int codigoPassageiro)

Propósito: Lista todos os voos de um passageiro

- Entrada: codigoPassageiro - Código do passageiro

- Funcionamento:

  - Busca todas as reservas

  - Exibe detalhes de cada voo

  - Mostra datas e status

void consultarPontosFidelidade(int codigoPassageiro)

Propósito: Consulta pontos de fidelidade

- Entrada: codigoPassageiro - Código do passageiro

- Funcionamento:

  - Busca saldo atual

  - Exibe histórico de pontos

  - Mostra status no programa

void atualizarPontosFidelidade(int codigoPassageiro)

Propósito: Atualiza pontos após nova reserva

- Entrada: codigoPassageiro - Código do passageiro

- Funcionamento:

  - Adiciona pontos da nova reserva

  - Atualiza saldo total

  - Verifica mudanças de status

int comparaStringsCase(const char *s1, const char *s2)

Propósito: Compara strings ignorando maiúsculas/minúsculas

- Entrada:

  - s1 - Primeira string

  - s2 - Segunda string

- Retorno: 1 se iguais, 0 se diferentes

- Funcionamento: Compara caractere a caractere ignorando case

TESTES

1. Teste do Módulo de Passageiros

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Código	Número inteiro maior que 0	Código aceito, prossegue cadastro	Números negativos, zero, letras, caracteres especiais	Erro: código inválido
Nome	Entre 3 e 50 caracteres, apenas letras e espaços	Nome aceito, prossegue cadastro	Menos de 3 caracteres, mais de 50, números, caracteres especiais	Erro: nome inválido
Endereço	Entre 5 e 100 caracteres, contendo letras e números	Endereço aceito, prossegue cadastro	Menos de 5 caracteres, mais de 100, sem número ou sem letras	Erro: endereço inválido
Telefone	Exatamente 11 dígitos numéricos	Telefone aceito, prossegue cadastro	Menos ou mais que 11 dígitos, letras, caracteres especiais	Erro: telefone inválido
Fidelidade	0 (não) ou 1 (sim)	Status aceito, finaliza cadastro	Qualquer outro valor	Erro: opção inválida

2. Teste do Módulo de Voos

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Código do Voo	Número inteiro maior que 0	Código aceito, prossegue cadastro	Números negativos, zero, letras, caracteres especiais	Erro: código inválido
Data	Data futura no formato DD/MM/AAAA	Data aceita, prossegue cadastro	Data passada, formato inválido, data inexistente	Erro: data inválida
Hora	Formato HH:MM (00:00 a 23:59)	Hora aceita, prossegue cadastro	Formato inválido, hora inexistente	Erro: hora inválida
Origem/Destino	Entre 2 e 50 caracteres, apenas letras e espaços	Local aceito, prossegue cadastro	Menos de 2 caracteres, números, caracteres especiais	Erro: local inválido
Tarifa	Valor maior que zero	Tarifa aceita, prossegue cadastro	Zero, valores negativos	Erro: tarifa inválida

3. Teste do Módulo de Reservas

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Código Reserva	Número inteiro maior que 0	Código aceito, prossegue cadastro	Números negativos, zero, letras	Erro: código inválido
Código Passageiro	Passageiro cadastrado	Passageiro validado	Passageiro inexistente	Erro: passageiro inválido
Código Voo	Voo cadastrado e ativo	Voo validado	Voo inexistente ou inativo	Erro: voo inválido
Número Assento	Entre 1 e 50, assento livre	Assento reservado	Assento inexistente ou ocupado	Erro: assento indisponível

4. Teste do Módulo de Tripulação

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Código Tripulante	Número inteiro maior que 0	Código aceito	Números negativos, zero, letras	Erro: código inválido
Nome	Entre 3 e 50 caracteres, apenas letras	Nome aceito	Menos de 3 caracteres, números	Erro: nome inválido
Telefone	10 ou 11 dígitos numéricos	Telefone aceito	Menos de 10 ou mais de 11 dígitos	Erro: telefone inválido
Cargo	"piloto", "copiloto", "comissario"	Cargo aceito	Qualquer outro valor	Erro: cargo inválido

5. Teste do Módulo de Pesquisa

Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Código/Nome Passageiro	Passageiro cadastrado	Dados exibidos	Passageiro não encontrado	Erro: passageiro não encontrado
Código/Nome Tripulação	Tripulante cadastrado	Dados exibidos	Tripulante não encontrado	Erro: tripulante não encontrado
Código Fidelidade	Passageiro cadastrado no programa	Pontos exibidos	Passageiro não participante	Erro: programa não encontrado

Código do Programa

Cadastro de passageiros .h e .c e testes automatizados

cadastrodepassageiros.h

```
#ifndef CADASTRAR_PASSAGEIRO_H
#define CADASTRAR_PASSAGEIRO_H
```

```h
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_PASSAGEIROS 100

typedef struct {
   int codigo;
   char nome[50];
   char endereco[100];
   char telefone[15];
   int fidelidade;
   int pontosFidelidade;
} Passageiro;

extern Passageiro listaPassageiros[MAX_PASSAGEIROS];
extern int qtdPassageiros;

// Funções de validação
int validarNome(const char *nome);
int validarEndereco(const char *endereco);
int validarTelefone(const char *telefone);
int validarNumeros(const char *str);

// Funções principais
void cadastrarPassageiro();
void listarPassageiros();

#endif
```

cadastrodepassageiros.c

```c
#include "cadastrarPassageiro.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

// Definição das variáveis globais
// Lista de passageiros e contador do total de passageiros cadastrados
Passageiro listaPassageiros[MAX_PASSAGEIROS];
int qtdPassageiros = 0;
```

```c
// Função para validar se uma string contém apenas números
int validarNumeros(const char *str) {
    if (str == NULL || *str == '\0') {
        return 0;  // Retorna falso para string nula ou vazia
    }
    while (*str) {
        if (!isdigit(*str)) {
            return 0;  // Retorna falso se algum caractere não for número
        }
        str++;
    }
    return 1;  // Retorna verdadeiro se todos os caracteres forem números
}

// Função para validar nomes
// Um nome válido deve ter entre 3 e 50 caracteres e conter apenas letras e espaços
int validarNome(const char *nome) {
    if (strlen(nome) < 3 || strlen(nome) > 50) return 0;

    for (int i = 0; nome[i]; i++) {
        if (!isalpha(nome[i]) && !isspace(nome[i])) {
            return 0;
        }
    }
    return 1;
}

// Função para validar endereços
// Um endereço válido deve ter entre 5 e 100 caracteres, contendo letras e números
int validarEndereco(const char *endereco) {
    if (strlen(endereco) < 5 || strlen(endereco) > 100) return 0;

    int temNumero = 0;
    int temLetra = 0;

    for (int i = 0; endereco[i]; i++) {
        if (isalpha(endereco[i])) temLetra = 1;
        if (isdigit(endereco[i])) temNumero = 1;
    }

    return temNumero && temLetra;
}
```

```c
// Função para validar telefones
// Um telefone válido deve conter exatamente 11 dígitos
int validarTelefone(const char *telefone) {
   if (strlen(telefone) != 11) return 0;

   for (int i = 0; telefone[i]; i++) {
       if (!isdigit(telefone[i])) {
           return 0;
       }
   }
   return 1;
}

// Função para limpar o buffer do teclado
// Usada para evitar problemas ao ler entradas do usuário
void limparBuffer() {
   int c;
   while ((c = getchar()) != '\n' && c != EOF);
}

// Função para cadastrar um novo passageiro
void cadastrarPassageiro() {
   if (qtdPassageiros >= MAX_PASSAGEIROS) {
       printf("Erro: Limite máximo de passageiros atingido.\n");
       return;
   }

   Passageiro novoPassageiro;
   novoPassageiro.pontosFidelidade = 0;  // Inicializa pontos

   printf("\n--- Cadastro de Passageiro ---\n");

   // Validação do código do passageiro
   while (1) {
       printf("Digite o código (somente números maiores que 0): ");
       if (scanf("%d", &novoPassageiro.codigo) != 1 || novoPassageiro.codigo <= 0) {
           printf("Erro: Código inválido!\n");
           limparBuffer();
           continue;
       }
```

```
        // Verifica se o código já está cadastrado
        int codigoExiste = 0;
        for (int i = 0; i < qtdPassageiros; i++) {
            if (listaPassageiros[i].codigo == novoPassageiro.codigo) {
                codigoExiste = 1;
                break;
            }
        }

        if (codigoExiste) {
            printf("Erro: Código já cadastrado!\n");
            continue;
        }
        break;
    }
    limparBuffer();

    // Validação do nome do passageiro
    while (1) {
        printf("Digite o nome: ");
        if (fgets(novoPassageiro.nome, sizeof(novoPassageiro.nome), stdin)) {
            novoPassageiro.nome[strcspn(novoPassageiro.nome, "\n")] = 0;
            if (validarNome(novoPassageiro.nome)) break;
        }
        printf("Erro: Nome inválido! Use apenas letras e espaços (3-50
caracteres).\n");
    }

    // Validação do endereço do passageiro
    while (1) {
        printf("Digite o endereço: ");
        if (fgets(novoPassageiro.endereco, sizeof(novoPassageiro.endereco), stdin)) {
            novoPassageiro.endereco[strcspn(novoPassageiro.endereco, "\n")] = 0;
            if (validarEndereco(novoPassageiro.endereco)) break;
        }
        printf("Erro: Endereço inválido! Inclua número e nome da rua (5-100
caracteres).\n");
    }

    // Validação do telefone do passageiro
    while (1) {
        printf("Digite o telefone (11 dígitos, apenas números): ");
```

```
        if (fgets(novoPassageiro.telefone, sizeof(novoPassageiro.telefone), stdin)) {
            novoPassageiro.telefone[strcspn(novoPassageiro.telefone, "\n")] = 0;
            if (validarTelefone(novoPassageiro.telefone)) break;
        }
        printf("Erro: Telefone inválido! Use exatamente 11 números.\n");
    }

    // Validação da participação no programa de fidelidade
    while (1) {
        printf("Participa do programa de fidelidade? (1 = Sim, 0 = Não): ");
        char input[10];
        if (fgets(input, sizeof(input), stdin)) {
            input[strcspn(input, "\n")] = 0;
            if (strlen(input) == 1 && (input[0] == '0' || input[0] == '1')) {
                novoPassageiro.fidelidade = input[0] - '0';
                break;
            }
        }
        printf("Erro: Digite apenas 0 ou 1!\n");
    }

    // Adiciona o passageiro à lista
    listaPassageiros[qtdPassageiros++] = novoPassageiro;
    printf("\nPassageiro cadastrado com sucesso!\n");
    printf("==============================\n");
}

// Função para listar todos os passageiros cadastrados
void listarPassageiros() {
    if (qtdPassageiros == 0) {
        printf("\nNenhum passageiro cadastrado.\n");
        return;
    }

    printf("\n=== Lista de Passageiros ===\n");
    for (int i = 0; i < qtdPassageiros; i++) {
        printf("\nPassageiro %d:\n", i+1);
        printf("Código: %d\n", listaPassageiros[i].codigo);
        printf("Nome: %s\n", listaPassageiros[i].nome);
        printf("Endereço: %s\n", listaPassageiros[i].endereco);
        printf("Telefone: %s\n", listaPassageiros[i].telefone);
        printf("Fidelidade: %s\n", listaPassageiros[i].fidelidade ? "Sim" : "Não");
```

```
        if (listaPassageiros[i].fidelidade) {
            printf("Pontos: %d\n", listaPassageiros[i].pontosFidelidade);
        }
        printf("----------------------------\n");
    }
}
```

testpassageiro.c

```
// Incluímos as bibliotecas necessárias

#include "../lib/munit/munit.h"    // Framework de testes

#include "../src/cadastrarPassageiro.h"  // Nossa implementação



/* Função de preparação (setup)

  Esta função é executada antes de cada teste para garantir um ambiente limpo */

static void* test_setup(const MunitParameter params[], void* user_data) {

    (void) params;

    (void) user_data;

    qtdPassageiros = 0;  // Reseta o contador de passageiros

    memset(listaPassageiros, 0, sizeof(listaPassageiros));  // Limpa o array

    return NULL;

}



/* Teste da função validarNumeros()

  Verifica se a validação de números está funcionando corretamente */

static MunitResult test_validar_numeros(const MunitParameter params[], void* data) {

    (void) params;

    (void) data;
```

```
    // Teste com entrada válida (apenas números)
    munit_assert_true(validarNumeros("123456789"));

    // Teste com entrada inválida (contém letras)
    munit_assert_false(validarNumeros("123abc456"));

    // Teste com entrada vazia
    munit_assert_false(validarNumeros(""));

    return MUNIT_OK;
}

/* Teste da função validarEndereco()
   Verifica se a validação de endereço está funcionando corretamente */
static MunitResult test_validar_endereco(const MunitParameter params[], void* data) {
    (void) params;
    (void) data;

    // Teste com endereço válido
    munit_assert_true(validarEndereco("Rua das Flores, 123"));

    // Teste com endereço vazio (deve falhar)
    munit_assert_false(validarEndereco(""));
```

```c
    return MUNIT_OK;
}

/* Teste do cadastro de passageiro
   Verifica se é possível cadastrar um passageiro com sucesso */
static MunitResult test_cadastro_passageiro(const MunitParameter params[], void* data)
{
    (void) params;
    (void) data;

    // Preparar dados de teste
    Passageiro p = {
        .codigo = 1,
        .nome = "João Silva",
        .endereco = "Rua ABC, 123",
        .telefone = "31999999999",
        .fidelidade = 1
    };

    // Simular o cadastro (adicionando diretamente para teste)
    listaPassageiros[qtdPassageiros++] = p;

    // Verificar se o passageiro foi cadastrado corretamente
```

```
    munit_assert_int(qtdPassageiros, ==, 1);

    munit_assert_int(listaPassageiros[0].codigo, ==, p.codigo);

    munit_assert_string_equal(listaPassageiros[0].nome, p.nome);



    return MUNIT_OK;

}



/* Definição da suite de testes

  Aqui listamos todos os testes que queremos executar */

static MunitTest test_suite_tests[] = {

    {

        "/validar_numeros",  // Nome do teste

        test_validar_numeros,  // Função de teste

        test_setup,  // Função de preparação

        NULL,  // Função de limpeza (não necessária aqui)

        MUNIT_TEST_OPTION_NONE,

        NULL

    },

    {

        "/validar_endereco",

        test_validar_endereco,

        test_setup,

        NULL,

        MUNIT_TEST_OPTION_NONE,
```

```
        NULL
    },
    {
        "/cadastro_passageiro",
        test_cadastro_passageiro,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
    { NULL, NULL, NULL, NULL, MUNIT_TEST_OPTION_NONE, NULL }  // Marca o fim da suite
};

/* Definição da suite principal */
static const MunitSuite test_suite = {
    "/passageiro_tests",  // Prefixo para todos os testes
    test_suite_tests,     // Array de testes
    NULL,                 // Suites filhas
    1,                    // Iterações
    MUNIT_SUITE_OPTION_NONE
};

/* Função principal que executa os testes */
int main(int argc, char* argv[MUNIT_ARRAY_PARAM(argc + 1)]) {
```

```
    return munit_suite_main(&test_suite, NULL, argc, argv);

}
```

Cadastro de tripulação .h e .c e testes automatizados

cadastrarTripulacao.h

```
#ifndef CADASTRAR_TRIPULACAO_H

#define CADASTRAR_TRIPULACAO_H



#include <stdio.h>

#include <string.h>

#include <ctype.h>



#define MAX_TRIPULACAO 50



typedef struct {

    int codigo;

    char nome[50];

    char telefone[15];

    char cargo[20];  // piloto, copiloto, comissário

} Tripulacao;



extern Tripulacao listaTripulacao[MAX_TRIPULACAO];

extern int qtdTripulacao;
```

```c
// Funções de validação
int validarCargoTripulacao(const char *cargo);
int validarNomeTripulacao(const char *nome);
int validarTelefoneTripulacao(const char *telefone);

// Funções principais
void cadastrarTripulacao();
void listarTripulacao();
int verificarTripulacaoMinima(int codigoPiloto, int codigoCopiloto);

#endif
```

cadastrarTripulacao.c

```c
#include "cadastrarTripulacao.h"
#include <stdlib.h>

Tripulacao listaTripulacao[MAX_TRIPULACAO];
int qtdTripulacao = 0;

void limparBufferTripulacao() {
   int c;
   while ((c = getchar()) != '\n' && c != EOF);
}
```

```c
int validarCargoTripulacao(const char *cargo) {
    char cargoMin[20];
    strncpy(cargoMin, cargo, sizeof(cargoMin) - 1);
    cargoMin[sizeof(cargoMin) - 1] = '\0';

    // Converte para minúsculas
    for (int i = 0; cargoMin[i]; i++) {
        cargoMin[i] = tolower(cargoMin[i]);
    }

    return (strcmp(cargoMin, "piloto") == 0 ||
            strcmp(cargoMin, "copiloto") == 0 ||
            strcmp(cargoMin, "comissario") == 0);
}

int validarNomeTripulacao(const char *nome) {
    if (strlen(nome) < 3 || strlen(nome) > 50) return 0;

    for (int i = 0; nome[i]; i++) {
        if (!isalpha(nome[i]) && !isspace(nome[i])) {
            return 0;
        }
    }
```

```c
    return 1;
}

int validarTelefoneTripulacao(const char *telefone) {
    if (telefone == NULL) {
        return 0;  // Return false if the phone number is NULL
    }
    int len = strlen(telefone);
    if (len != 10 && len != 11) {
        return 0;  // Return false if the phone number is not 10 or 11 digits long
    }
    for (int i = 0; i < len; i++) {
        if (!isdigit(telefone[i])) {
            return 0;  // Return false if any character is not a digit
        }
    }
    return 1;  // Return true if all characters are digits and the length is 10 or 11
}

void cadastrarTripulacao() {
    if (qtdTripulacao >= MAX_TRIPULACAO) {
        printf("Erro: Limite máximo de tripulantes atingido.\n");
        return;
    }
```

```c
Tripulacao novoTripulante;

printf("\n--- Cadastro de Tripulação ---\n");

// Validação do código
while (1) {
    printf("Digite o código (somente números maiores que 0): ");
    if (scanf("%d", &novoTripulante.codigo) != 1 || novoTripulante.codigo <= 0) {
        printf("Erro: Código inválido!\n");
        limparBufferTripulacao();
        continue;
    }

    int codigoExiste = 0;
    for (int i = 0; i < qtdTripulacao; i++) {
        if (listaTripulacao[i].codigo == novoTripulante.codigo) {
            codigoExiste = 1;
            break;
        }
    }

    if (codigoExiste) {
        printf("Erro: Código já cadastrado!\n");
```

```
            continue;

        }

        break;

    }

    limparBufferTripulacao();



    // Validação do nome

    while (1) {

        printf("Digite o nome: ");

        if (fgets(novoTripulante.nome, sizeof(novoTripulante.nome), stdin)) {

            novoTripulante.nome[strcspn(novoTripulante.nome, "\n")] = 0;

            if (validarNomeTripulacao(novoTripulante.nome)) break;

        }

        printf("Erro: Nome inválido! Use apenas letras e espaços (3-50
caracteres).\n");

    }



    // Validação do telefone

    while (1) {

        printf("Digite o telefone (11 dígitos, apenas números): ");

        if (fgets(novoTripulante.telefone, sizeof(novoTripulante.telefone), stdin)) {

            novoTripulante.telefone[strcspn(novoTripulante.telefone, "\n")] = 0;

            if (validarTelefoneTripulacao(novoTripulante.telefone)) break;

        }
```

```
        printf("Erro: Telefone inválido! Use exatamente 11 números.\n");
    }

    // Validação do cargo
    while (1) {
        printf("Digite o cargo (piloto, copiloto ou comissario): ");
        if (fgets(novoTripulante.cargo, sizeof(novoTripulante.cargo), stdin)) {
            novoTripulante.cargo[strcspn(novoTripulante.cargo, "\n")] = 0;
            if (validarCargoTripulacao(novoTripulante.cargo)) break;
        }
        printf("Erro: Cargo inválido! Use apenas: piloto, copiloto ou comissario.\n");
    }

    listaTripulacao[qtdTripulacao++] = novoTripulante;
    printf("\nTripulante cadastrado com sucesso!\n");
    printf("============================\n");
}

void listarTripulacao() {
    if (qtdTripulacao == 0) {
        printf("\nNenhum tripulante cadastrado.\n");
        return;
    }
```

```
    printf("\n=== Lista de Tripulantes ===\n");

    for (int i = 0; i < qtdTripulacao; i++) {

        printf("\nTripulante %d:\n", i+1);

        printf("Código: %d\n", listaTripulacao[i].codigo);

        printf("Nome: %s\n", listaTripulacao[i].nome);

        printf("Cargo: %s\n", listaTripulacao[i].cargo);

        printf("Telefone: %s\n", listaTripulacao[i].telefone);

        printf("----------------------------\n");

    }

}



int verificarTripulacaoMinima(int codigoPiloto, int codigoCopiloto) {

    int pilotoEncontrado = 0, copilotoEncontrado = 0;



    for (int i = 0; i < qtdTripulacao; i++) {

        if (listaTripulacao[i].codigo == codigoPiloto &&

            strcmp(listaTripulacao[i].cargo, "piloto") == 0) {

            pilotoEncontrado = 1;

        }

        if (listaTripulacao[i].codigo == codigoCopiloto &&

            strcmp(listaTripulacao[i].cargo, "copiloto") == 0) {

            copilotoEncontrado = 1;

        }

    }
```

```
    return pilotoEncontrado && copilotoEncontrado;
}
```

test_tripulacao.c

```
#include "../lib/munit/munit.h"
#include "../src/cadastrarTripulacao.h"


/* Função de preparação (setup)
   Esta função é executada antes de cada teste para garantir um ambiente limpo */
static void* test_setup(const MunitParameter params[], void* user_data) {
    (void) params;
    (void) user_data;
    qtdTripulacao = 0;  // Reseta o contador
    memset(listaTripulacao, 0, sizeof(listaTripulacao));  // Limpa a lista
    return NULL;
}


/* Teste de validação de cargo
   Verifica se a função aceita apenas os cargos permitidos */
static MunitResult test_validar_cargo(const MunitParameter params[], void* data) {
    (void) params;
    (void) data;
```

```c
    // Teste com cargos válidos
    munit_assert_true(validarCargoTripulacao("piloto"));
    munit_assert_true(validarCargoTripulacao("copiloto"));
    munit_assert_true(validarCargoTripulacao("comissario"));

    // Teste com variações de maiúsculas/minúsculas
    munit_assert_true(validarCargoTripulacao("PILOTO"));
    munit_assert_true(validarCargoTripulacao("Copiloto"));

    // Teste com cargos inválidos
    munit_assert_false(validarCargoTripulacao("engenheiro"));
    munit_assert_false(validarCargoTripulacao(""));

    return MUNIT_OK;
}

/* Teste de validação de telefone
   Verifica se a função aceita apenas números e respeita o tamanho máximo */
static MunitResult test_validar_telefone(const MunitParameter params[], void* data) {
    (void) params;
    (void) data;

    // Teste com telefones válidos
    munit_assert_true(validarTelefoneTripulacao("1234567890"));
```

```
    munit_assert_true(validarTelefoneTripulacao("31999999999"));

    // Teste com telefones inválidos
    munit_assert_false(validarTelefoneTripulacao("123-456-789")); // Contém caracteres especiais
    munit_assert_false(validarTelefoneTripulacao("")); // Vazio
    munit_assert_false(validarTelefoneTripulacao("1234567890123456")); // Muito longo

    return MUNIT_OK;
}

/* Teste de cadastro de tripulação
   Verifica se é possível cadastrar um membro da tripulação corretamente */
static MunitResult test_cadastro_tripulacao(const MunitParameter params[], void* data) {
    (void) params;
    (void) data;

    // Preparar dados de teste
    Tripulacao novo = {
        .codigo = 1,
        .nome = "João Silva",
        .telefone = "31999999999",
        .cargo = "piloto"
    };
```

```c
    // Simular cadastro
    listaTripulacao[qtdTripulacao++] = novo;

    // Verificações
    munit_assert_int(qtdTripulacao, ==, 1);
    munit_assert_int(listaTripulacao[0].codigo, ==, novo.codigo);
    munit_assert_string_equal(listaTripulacao[0].nome, novo.nome);
    munit_assert_string_equal(listaTripulacao[0].cargo, novo.cargo);

    return MUNIT_OK;
}

/* Definição da suite de testes
  Aqui listamos todos os testes que criamos */
static MunitTest test_suite_tests[] = {
    {
        "/validar_cargo",
        test_validar_cargo,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
```

```
    {
        "/validar_telefone",
        test_validar_telefone,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
    {
        "/cadastro_tripulacao",
        test_cadastro_tripulacao,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
    { NULL, NULL, NULL, NULL, MUNIT_TEST_OPTION_NONE, NULL }
};


/* Definição da suite principal */
static const MunitSuite test_suite = {
    "/tripulacao",
    test_suite_tests,
    NULL,
```

```
    1,
    MUNIT_SUITE_OPTION_NONE
};

/* Função principal */
int main(int argc, char* argv[MUNIT_ARRAY_PARAM(argc + 1)]) {
    return munit_suite_main(&test_suite, NULL, argc, argv);
}
```

Cadastro de Voo .h e .c e testes automatizados

cadastrarVoo.h

```
#ifndef CADASTRAR_VOO_H
#define CADASTRAR_VOO_H

#include <stdio.h>
#include <string.h>
#include <time.h>
#include "cadastrarTripulacao.h"

#define MAX_VOOS 100
#define MAX_ASSENTOS 50

typedef struct {
```

```c
  int codigo;
  char data[11];      // DD/MM/AAAA
  char hora[6];       // HH:MM
  char origem[50];
  char destino[50];
  int codigoAviao;
  int codigoPiloto;
  int codigoCopiloto;
  int codigoComissario;
  int status;         // 1 = ativo, 0 = inativo
  float tarifa;
} Voo;

typedef struct {
  int numero;
  int codigoVoo;
  int status;         // 1 = ocupado, 0 = livre
} Assento;

extern Voo listaVoos[MAX_VOOS];
extern Assento listaAssentos[MAX_VOOS][MAX_ASSENTOS];
extern int qtdVoos;

// Funções de validação
```

```c
int validarData(const char *data);

int validarHora(const char *hora);

int validarTarifa(float tarifa);

int validarTripulacaoCompleta(int codigoPiloto, int codigoCopiloto);



// Funções principais

void cadastrarVoo();

void listarVoos();

void configurarAssentos(int indiceVoo);

int verificarStatusVoo(int codigoVoo);



#endif
```

cadastrarVoo.c

```c
#include "cadastrarVoo.h"
#include <stdlib.h>
#include <ctype.h>

Voo listaVoos[MAX_VOOS];
Assento listaAssentos[MAX_VOOS][MAX_ASSENTOS];
int qtdVoos = 0;

void limparBufferVoo() {
   int c;
   while ((c = getchar()) != '\n' && c != EOF);
}

int validarData(const char *data) {
   if (strlen(data) != 10 || data[2] != '/' || data[5] != '/') return 0;

   int dia, mes, ano;
   if (sscanf(data, "%d/%d/%d", &dia, &mes, &ano) != 3) return 0;
```

```c
    if (ano < 2024) return 0;  // Não aceita datas passadas
    if (mes < 1 || mes > 12) return 0;

    int diasPorMes[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
    if ((ano % 4 == 0 && ano % 100 != 0) || (ano % 400 == 0)) diasPorMes[2] = 29;  // Ano bissexto

    return dia > 0 && dia <= diasPorMes[mes];
}

int validarHora(const char *hora) {
    // Verifica o tamanho e o formato básico
    if (strlen(hora) != 5) return 0;
    if (hora[2] != ':') return 0;

    // Verifica se são números
    if (!isdigit(hora[0]) || !isdigit(hora[1]) ||
        !isdigit(hora[3]) || !isdigit(hora[4])) {
        return 0;
    }

    // Converte e valida horas e minutos
    int h = (hora[0] - '0') * 10 + (hora[1] - '0');
    int m = (hora[3] - '0') * 10 + (hora[4] - '0');

    // Verifica faixa de valores
    if (h < 0 || h > 23) return 0;
    if (m < 0 || m > 59) return 0;

    return 1;
}

int validarTarifa(float tarifa) {
    return tarifa > 0.0;
}

int validarLocalizacao(const char *local) {
    if (strlen(local) < 2 || strlen(local) > 50) return 0;

    for (int i = 0; local[i]; i++) {
        if (!isalpha(local[i]) && !isspace(local[i])) {
```

```
            return 0;
        }
    }
    return 1;
}

void cadastrarVoo() {
    if (qtdVoos >= MAX_VOOS) {
        printf("Erro: Limite máximo de voos atingido.\n");
        return;
    }

    Voo novoVoo;

    printf("\n--- Cadastro de Voo ---\n");

    // Validação do código do voo
    while (1) {
        printf("Digite o código do voo: ");
        if (scanf("%d", &novoVoo.codigo) != 1 || novoVoo.codigo <= 0) {
            printf("Erro: Código inválido!\n");
            limparBufferVoo();
            continue;
        }

        int codigoExiste = 0;
        for (int i = 0; i < qtdVoos; i++) {
            if (listaVoos[i].codigo == novoVoo.codigo) {
                codigoExiste = 1;
                break;
            }
        }

        if (codigoExiste) {
            printf("Erro: Código de voo já existe!\n");
            continue;
        }
        break;
    }
    limparBufferVoo();

    // Validação da data
```

```
    while (1) {
        printf("Digite a data (DD/MM/AAAA): ");
        if (fgets(novoVoo.data, sizeof(novoVoo.data), stdin)) {
            novoVoo.data[strcspn(novoVoo.data, "\n")] = 0;
            if (validarData(novoVoo.data)) break;
        }
        printf("Erro: Data inválida! Use o formato DD/MM/AAAA e data futura.\n");
    }

limparBufferVoo();
    // Validação da hora
    while (1) {
        printf("Digite a hora (HH:MM): ");
        if (fgets(novoVoo.hora, sizeof(novoVoo.hora), stdin)) {
            novoVoo.hora[strcspn(novoVoo.hora, "\n")] = 0;
            if (validarHora(novoVoo.hora)) break;
        }
        printf("Erro: Hora inválida! Use o formato HH:MM (24h).\n");
    }

limparBufferVoo();
    // Validação da origem
    while (1) {
        printf("Digite a origem: ");
        if (fgets(novoVoo.origem, sizeof(novoVoo.origem), stdin)) {
            novoVoo.origem[strcspn(novoVoo.origem, "\n")] = 0;
            if (validarLocalizacao(novoVoo.origem)) break;
        }
        printf("Erro: Origem inválida! Use apenas letras e espaços.\n");
    }

    // Validação do destino
    while (1) {
        printf("Digite o destino: ");
        if (fgets(novoVoo.destino, sizeof(novoVoo.destino), stdin)) {
            novoVoo.destino[strcspn(novoVoo.destino, "\n")] = 0;
            if (validarLocalizacao(novoVoo.destino)) break;
        }
        printf("Erro: Destino inválido! Use apenas letras e espaços.\n");
    }

    // Validação da tarifa
```

```c
    while (1) {
        printf("Digite a tarifa: R$ ");
        if (scanf("%f", &novoVoo.tarifa) == 1 && validarTarifa(novoVoo.tarifa)) {
            break;
        }
        printf("Erro: Tarifa inválida! Digite um valor maior que zero.\n");
        limparBufferVoo();
    }
    limparBufferVoo();

    // Validação da tripulação
    printf("\nCadastro da Tripulação do Voo\n");
    while (1) {
        printf("Digite o código do piloto: ");
        scanf("%d", &novoVoo.codigoPiloto);
        printf("Digite o código do copiloto: ");
        scanf("%d", &novoVoo.codigoCopiloto);

        if (verificarTripulacaoMinima(novoVoo.codigoPiloto, novoVoo.codigoCopiloto)) {
            break;
        }
        printf("Erro: Tripulação inválida! Verifique os códigos do piloto e
copiloto.\n");
    }

    printf("Digite o código do comissário: ");
    scanf("%d", &novoVoo.codigoComissario);
    limparBufferVoo();

    // Configura o status inicial como ativo
    novoVoo.status = 1;

    // Salva o voo e configura os assentos
    listaVoos[qtdVoos] = novoVoo;
    configurarAssentos(qtdVoos);
    qtdVoos++;

    printf("\nVoo cadastrado com sucesso!\n");
    printf("============================\n");
}

void configurarAssentos(int indiceVoo) {
```

```
    for (int i = 0; i < MAX_ASSENTOS; i++) {
        listaAssentos[indiceVoo][i].numero = i + 1;
        listaAssentos[indiceVoo][i].codigoVoo = listaVoos[indiceVoo].codigo;
        listaAssentos[indiceVoo][i].status = 0;  // Livre
    }
}

void listarVoos() {
    if (qtdVoos == 0) {
        printf("\nNenhum voo cadastrado.\n");
        return;
    }

    printf("\n=== Lista de Voos ===\n");
    for (int i = 0; i < qtdVoos; i++) {
        printf("\nVoo %d:\n", i+1);
        printf("Código: %d\n", listaVoos[i].codigo);
        printf("Data: %s\n", listaVoos[i].data);
        printf("Hora: %s\n", listaVoos[i].hora);
        printf("Origem: %s\n", listaVoos[i].origem);
        printf("Destino: %s\n", listaVoos[i].destino);
        printf("Tarifa: R$ %.2f\n", listaVoos[i].tarifa);
        printf("Status: %s\n", listaVoos[i].status ? "Ativo" : "Inativo");
        printf("----------------------------\n");
    }
}

int verificarStatusVoo(int codigoVoo) {
    for (int i = 0; i < qtdVoos; i++) {
        if (listaVoos[i].codigo == codigoVoo) {
            return listaVoos[i].status;
        }
    }
    return 0;  // Voo não encontrado
}
```

Cadastro de Reservas .h e .c e testes automatizados

cadastrarReserva.h

```
#ifndef CADASTRAR_RESERVA_H
```

```c
#define CADASTRAR_RESERVA_H



#include <stdio.h>

#include <string.h>

#include "cadastrarPassageiro.h"

#include "cadastrarVoo.h"



#define MAX_RESERVAS 1000

#define PONTOS_POR_VOO 10



typedef struct {

  int codigo;

  int codigoVoo;

  int numeroAssento;

  int codigoPassageiro;

  float valorTotal;

} Reserva;



extern Reserva listaReservas[MAX_RESERVAS];

extern int qtdReservas;



// Funções de validação

int verificarDisponibilidadeAssento(int codigoVoo, int numeroAssento);

int validarPassageiroExiste(int codigoPassageiro);
```

```c
int validarVooAtivo(int codigoVoo);



// Funções principais

void cadastrarReserva();

void baixarReserva();

void listarReservasPassageiro(int codigoPassageiro);

void atualizarPontosFidelidade(int codigoPassageiro);



#endif
```

cadastrarReserva.c

```c
#include "../lib/munit/munit.h"

#include "../src/cadastrarReserva.h"



#define UNUSED(x) (void)(x)



/* Função de preparação que limpa os dados antes de cada teste */

static void* test_setup(const MunitParameter params[], void* user_data) {

    UNUSED(params);

    UNUSED(user_data);



    // Limpa todas as estruturas de dados

    qtdReservas = 0;

    qtdVoos = 0;
```

```c
    qtdPassageiros = 0;

    memset(listaReservas, 0, sizeof(listaReservas));

    memset(listaVoos, 0, sizeof(listaVoos));

    memset(listaPassageiros, 0, sizeof(listaPassageiros));

    memset(listaAssentos, 0, sizeof(listaAssentos));



    return NULL;

}



/* Teste da verificação de disponibilidade de assento */

static MunitResult test_verificar_disponibilidade_assento(const MunitParameter
params[], void* data) {

    UNUSED(params);

    UNUSED(data);



    // Configura um voo de teste

    Voo vooTeste = {

        .codigo = 1,

        .status = 1

    };

    listaVoos[qtdVoos++] = vooTeste;



    // Configura assentos iniciais como livres

    for (int i = 0; i < MAX_ASSENTOS; i++) {
```

```
        listaAssentos[0][i].status = 0;
    }

    // Testa assento válido e livre
    munit_assert_true(verificarDisponibilidadeAssento(1, 1));

    // Ocupa um assento e testa novamente
    listaAssentos[0][0].status = 1;
    munit_assert_false(verificarDisponibilidadeAssento(1, 1));

    // Testa assento inválido
    munit_assert_false(verificarDisponibilidadeAssento(1, MAX_ASSENTOS + 1));
    munit_assert_false(verificarDisponibilidadeAssento(1, 0));

    // Testa voo inexistente
    munit_assert_false(verificarDisponibilidadeAssento(999, 1));

    return MUNIT_OK;
}

/* Teste de validação de passageiro */
static MunitResult test_validar_passageiro_existe(const MunitParameter params[], void*
data) {
    UNUSED(params);
```

```c
    UNUSED(data);

    // Configura um passageiro de teste
    Passageiro passageiroTeste = {
        .codigo = 1,
        .fidelidade = 1
    };
    listaPassageiros[qtdPassageiros++] = passageiroTeste;

    // Testa passageiro existente
    munit_assert_true(validarPassageiroExiste(1));

    // Testa passageiro inexistente
    munit_assert_false(validarPassageiroExiste(999));

    return MUNIT_OK;
}

/* Teste de cadastro de reserva */
static MunitResult test_cadastro_reserva(const MunitParameter params[], void* data) {
    UNUSED(params);
    UNUSED(data);

    // Configura dados necessários para o teste
```

```
Voo vooTeste = {
    .codigo = 1,
    .status = 1,
    .tarifa = 500.0
};
listaVoos[qtdVoos++] = vooTeste;

Passageiro passageiroTeste = {
    .codigo = 1,
    .fidelidade = 1
};
listaPassageiros[qtdPassageiros++] = passageiroTeste;

// Cria uma reserva manualmente (simulando input do usuário)
Reserva novaReserva = {
    .codigo = 1,
    .codigoVoo = 1,
    .numeroAssento = 1,
    .codigoPassageiro = 1,
    .valorTotal = 500.0
};
listaReservas[qtdReservas++] = novaReserva;

// Verifica se a reserva foi criada corretamente
```

```
    munit_assert_int(qtdReservas, ==, 1);

    munit_assert_int(listaReservas[0].codigo, ==, 1);

    munit_assert_float(listaReservas[0].valorTotal, ==, 500.0);



    return MUNIT_OK;

}



/* Lista de testes */

static MunitTest test_suite_tests[] = {

    {

        "/verificar_disponibilidade_assento",

        test_verificar_disponibilidade_assento,

        test_setup,

        NULL,

        MUNIT_TEST_OPTION_NONE,

        NULL

    },

    {

        "/validar_passageiro_existe",

        test_validar_passageiro_existe,

        test_setup,

        NULL,

        MUNIT_TEST_OPTION_NONE,

        NULL
```

```
    },
    {
        "/cadastro_reserva",
        test_cadastro_reserva,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
    { NULL, NULL, NULL, NULL, MUNIT_TEST_OPTION_NONE, NULL }
};

/* Definição da suite de testes */
static const MunitSuite test_suite = {
    "/reserva",
    test_suite_tests,
    NULL,
    1,
    MUNIT_SUITE_OPTION_NONE
};

int main(int argc, char* argv[MUNIT_ARRAY_PARAM(argc + 1)]) {
    return munit_suite_main(&test_suite, NULL, argc, argv);
}
```

test_reserva.c

```c
#include "../lib/munit/munit.h"

#include "../src/cadastrarReserva.h"



#define UNUSED(x) (void)(x)



/* Função de preparação que limpa os dados antes de cada teste */

static void* test_setup(const MunitParameter params[], void* user_data) {

    UNUSED(params);

    UNUSED(user_data);



    // Limpa todas as estruturas de dados

    qtdReservas = 0;

    qtdVoos = 0;

    qtdPassageiros = 0;

    memset(listaReservas, 0, sizeof(listaReservas));

    memset(listaVoos, 0, sizeof(listaVoos));

    memset(listaPassageiros, 0, sizeof(listaPassageiros));

    memset(listaAssentos, 0, sizeof(listaAssentos));



    return NULL;

}
```

```c
/* Teste da verificação de disponibilidade de assento */

static MunitResult test_verificar_disponibilidade_assento(const MunitParameter
params[], void* data) {

   UNUSED(params);

   UNUSED(data);



   // Configura um voo de teste

   Voo vooTeste = {

       .codigo = 1,

       .status = 1

   };

   listaVoos[qtdVoos++] = vooTeste;



   // Configura assentos iniciais como livres

   for (int i = 0; i < MAX_ASSENTOS; i++) {

       listaAssentos[0][i].status = 0;

   }



   // Testa assento válido e livre

   munit_assert_true(verificarDisponibilidadeAssento(1, 1));



   // Ocupa um assento e testa novamente

   listaAssentos[0][0].status = 1;

   munit_assert_false(verificarDisponibilidadeAssento(1, 1));
```

```
    // Testa assento inválido
    munit_assert_false(verificarDisponibilidadeAssento(1, MAX_ASSENTOS + 1));
    munit_assert_false(verificarDisponibilidadeAssento(1, 0));

    // Testa voo inexistente
    munit_assert_false(verificarDisponibilidadeAssento(999, 1));

    return MUNIT_OK;
}

/* Teste de validação de passageiro */
static MunitResult test_validar_passageiro_existe(const MunitParameter params[], void* data) {
    UNUSED(params);
    UNUSED(data);

    // Configura um passageiro de teste
    Passageiro passageiroTeste = {
        .codigo = 1,
        .fidelidade = 1
    };
    listaPassageiros[qtdPassageiros++] = passageiroTeste;
```

```
    // Testa passageiro existente
    munit_assert_true(validarPassageiroExiste(1));

    // Testa passageiro inexistente
    munit_assert_false(validarPassageiroExiste(999));

    return MUNIT_OK;
}

/* Teste de cadastro de reserva */
static MunitResult test_cadastro_reserva(const MunitParameter params[], void* data) {
    UNUSED(params);
    UNUSED(data);

    // Configura dados necessários para o teste
    Voo vooTeste = {
        .codigo = 1,
        .status = 1,
        .tarifa = 500.0
    };
    listaVoos[qtdVoos++] = vooTeste;

    Passageiro passageiroTeste = {
        .codigo = 1,
```

```
        .fidelidade = 1
    };
    listaPassageiros[qtdPassageiros++] = passageiroTeste;

    // Cria uma reserva manualmente (simulando input do usuário)
    Reserva novaReserva = {
        .codigo = 1,
        .codigoVoo = 1,
        .numeroAssento = 1,
        .codigoPassageiro = 1,
        .valorTotal = 500.0
    };
    listaReservas[qtdReservas++] = novaReserva;

    // Verifica se a reserva foi criada corretamente
    munit_assert_int(qtdReservas, ==, 1);
    munit_assert_int(listaReservas[0].codigo, ==, 1);
    munit_assert_float(listaReservas[0].valorTotal, ==, 500.0);

    return MUNIT_OK;
}

/* Lista de testes */
static MunitTest test_suite_tests[] = {
```

```
    {
        "/verificar_disponibilidade_assento",
        test_verificar_disponibilidade_assento,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
    {
        "/validar_passageiro_existe",
        test_validar_passageiro_existe,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
    {
        "/cadastro_reserva",
        test_cadastro_reserva,
        test_setup,
        NULL,
        MUNIT_TEST_OPTION_NONE,
        NULL
    },
```

```
    { NULL, NULL, NULL, NULL, MUNIT_TEST_OPTION_NONE, NULL }
};

/* Definição da suite de testes */
static const MunitSuite test_suite = {
    "/reserva",
    test_suite_tests,
    NULL,
    1,
    MUNIT_SUITE_OPTION_NONE
};

int main(int argc, char* argv[MUNIT_ARRAY_PARAM(argc + 1)]) {
    return munit_suite_main(&test_suite, NULL, argc, argv);
}
```

Pesquisa .h e .c

pesquisa.h

```
#ifndef PESQUISA_H
#define PESQUISA_H

#include "cadastrarPassageiro.h"
#include "cadastrarTripulacao.h"
#include "cadastrarVoo.h"
```

```c
#include "cadastrarReserva.h"



// Funções de pesquisa para passageiros

void pesquisarPassageiroPorCodigo(int codigo);

void pesquisarPassageiroPorNome(const char *nome);

void listarVoosPassageiro(int codigoPassageiro);



// Funções de pesquisa para tripulação

void pesquisarTripulacaoPorCodigo(int codigo);

void pesquisarTripulacaoPorNome(const char *nome);



// Funções para programa de fidelidade

void consultarPontosFidelidade(int codigoPassageiro);

void atualizarPontosFidelidade(int codigoPassageiro);



#endif
```

pesquisa.c

```c
#include "pesquisa.h"

#include <string.h>

#include <ctype.h>



// Funções auxiliares e de pesquisa já existentes...
```

```c
void atualizarPontosFidelidade(int codigoPassageiro) {
    for (int i = 0; i < qtdPassageiros; i++) {
        if (listaPassageiros[i].codigo == codigoPassageiro) {
            listaPassageiros[i].pontosFidelidade += PONTOS_POR_VOO;  // Incrementa os pontos
            printf("Pontos atualizados! Pontos atuais: %d\n", listaPassageiros[i].pontosFidelidade);
            return;
        }
    }
    printf("Passageiro não encontrado para atualizar os pontos.\n");
}

void consultarPontosFidelidade(int codigoPassageiro) {
    for (int i = 0; i < qtdPassageiros; i++) {
        if (listaPassageiros[i].codigo == codigoPassageiro) {
            printf("Pontos de fidelidade do passageiro %s: %d\n", listaPassageiros[i].nome, listaPassageiros[i].pontosFidelidade);
            return;
        }
    }
    printf("Passageiro não encontrado.\n");
}

void limparBufferPesquisa() {
```

```c
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

int comparaStringsCase(const char *s1, const char *s2) {
    if (!s1 || !s2) return 0;

    while (*s1 && *s2) {
        if (tolower(*s1) != tolower(*s2)) {
            return 0;
        }
        s1++;
        s2++;
    }
    return *s1 == *s2;
}

void pesquisarPassageiroPorCodigo(int codigo) {
    for (int i = 0; i < qtdPassageiros; i++) {
        if (listaPassageiros[i].codigo == codigo) {
            printf("\n=== Passageiro Encontrado ===\n");
            printf("Código: %d\n", listaPassageiros[i].codigo);
            printf("Nome: %s\n", listaPassageiros[i].nome);
            printf("Endereço: %s\n", listaPassageiros[i].endereco);
```

```
            printf("Telefone: %s\n", listaPassageiros[i].telefone);
            printf("Fidelidade: %s\n", listaPassageiros[i].fidelidade ? "Sim" : "Não");
            return;
        }
    }
    printf("Passageiro não encontrado!\n");
}

void pesquisarPassageiroPorNome(const char *nome) {
    int encontrou = 0;
    printf("\n=== Resultado da Pesquisa ===\n");

    for (int i = 0; i < qtdPassageiros; i++) {
        if (strstr(listaPassageiros[i].nome, nome) != NULL) {
            printf("\nPassageiro %d:\n", i+1);
            printf("Código: %d\n", listaPassageiros[i].codigo);
            printf("Nome: %s\n", listaPassageiros[i].nome);
            printf("Endereço: %s\n", listaPassageiros[i].endereco);
            printf("Telefone: %s\n", listaPassageiros[i].telefone);
            printf("Fidelidade: %s\n", listaPassageiros[i].fidelidade ? "Sim" : "Não");
            printf("----------------------------\n");
            encontrou = 1;
        }
    }
```

```c
    if (!encontrou) {
        printf("Nenhum passageiro encontrado com este nome!\n");
    }
}

void pesquisarTripulacaoPorCodigo(int codigo) {
    for (int i = 0; i < qtdTripulacao; i++) {
        if (listaTripulacao[i].codigo == codigo) {
            printf("\n=== Tripulante Encontrado ===\n");
            printf("Código: %d\n", listaTripulacao[i].codigo);
            printf("Nome: %s\n", listaTripulacao[i].nome);
            printf("Cargo: %s\n", listaTripulacao[i].cargo);
            printf("Telefone: %s\n", listaTripulacao[i].telefone);
            return;
        }
    }
    printf("Tripulante não encontrado!\n");
}

void pesquisarTripulacaoPorNome(const char *nome) {
    int encontrou = 0;
    printf("\n=== Resultado da Pesquisa ===\n");
```

```c
    for (int i = 0; i < qtdTripulacao; i++) {
        if (strstr(listaTripulacao[i].nome, nome) != NULL) {
            printf("\nTripulante %d:\n", i+1);
            printf("Código: %d\n", listaTripulacao[i].codigo);
            printf("Nome: %s\n", listaTripulacao[i].nome);
            printf("Cargo: %s\n", listaTripulacao[i].cargo);
            printf("Telefone: %s\n", listaTripulacao[i].telefone);
            printf("----------------------------\n");
            encontrou = 1;
        }
    }

    if (!encontrou) {
        printf("Nenhum tripulante encontrado com este nome!\n");
    }
}

void listarVoosPassageiro(int codigoPassageiro) {
    int encontrou = 0;
    printf("\n=== Voos do Passageiro ===\n");

    for (int i = 0; i < qtdReservas; i++) {
        if (listaReservas[i].codigoPassageiro == codigoPassageiro) {
            for (int j = 0; j < qtdVoos; j++) {
```

```
            if (listaVoos[j].codigo == listaReservas[i].codigoVoo) {
                printf("\nReserva: %d\n", listaReservas[i].codigo);
                printf("Voo: %d\n", listaVoos[j].codigo);
                printf("Data: %s\n", listaVoos[j].data);
                printf("Hora: %s\n", listaVoos[j].hora);
                printf("Origem: %s\n", listaVoos[j].origem);
                printf("Destino: %s\n", listaVoos[j].destino);
                printf("Assento: %d\n", listaReservas[i].numeroAssento);
                printf("----------------------------\n");
                encontrou = 1;
            }
        }
    }

    if (!encontrou) {
        printf("Nenhum voo encontrado para este passageiro!\n");
    }
}
```

Arquivo principal main.c

main.c

```
#include <stdio.h>
```

```c
#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include "cadastrarPassageiro.h"

#include "cadastrarTripulacao.h"

#include "cadastrarVoo.h"

#include "cadastrarReserva.h"

#include "pesquisa.h"



void limparBufferMain() {

 int c;

 while ((c = getchar()) != '\n' && c != EOF);

}



void menuPesquisa() {

 int opcao;

 printf("\n--- Menu de Pesquisa ---\n");

 printf("1. Pesquisar Passageiro por Código\n");

 printf("2. Pesquisar Passageiro por Nome\n");

 printf("3. Pesquisar Tripulação por Código\n");

 printf("4. Pesquisar Tripulação por Nome\n");

 printf("5. Listar Voos de um Passageiro\n");

 printf("6. Voltar\n");

 printf("Escolha uma opção: ");
```

```c
scanf("%d", &opcao);
limparBufferMain();

int codigo;
char nome[50];

switch(opcao) {
    case 1:
        printf("Digite o código do passageiro: ");
        scanf("%d", &codigo);
        limparBufferMain();
        pesquisarPassageiroPorCodigo(codigo);
        break;

    case 2:
        printf("Digite o nome do passageiro: ");
        fgets(nome, sizeof(nome), stdin);
        nome[strcspn(nome, "\n")] = 0;
        pesquisarPassageiroPorNome(nome);
        break;

    case 3:
        printf("Digite o código do tripulante: ");
```

```
        scanf("%d", &codigo);
        limparBufferMain();
        pesquisarTripulacaoPorCodigo(codigo);
        break;

    case 4:
        printf("Digite o nome do tripulante: ");
        fgets(nome, sizeof(nome), stdin);
        nome[strcspn(nome, "\n")] = 0;
        pesquisarTripulacaoPorNome(nome);
        break;

    case 5:
        printf("Digite o código do passageiro: ");
        scanf("%d", &codigo);
        limparBufferMain();
        listarVoosPassageiro(codigo);
        break;

    case 6:
        return;

    default:
        printf("Opção inválida!\n");
```

```c
 }

}



int main() {

 int opcao;



 while (1) {

     printf("\n=== Sistema de Gerenciamento de Voos ===\n");

     printf("1. Cadastrar Passageiro\n");

     printf("2. Listar Passageiros\n");

     printf("3. Cadastrar Tripulação\n");

     printf("4. Listar Tripulação\n");

     printf("5. Cadastrar Voo\n");

     printf("6. Listar Voos\n");

     printf("7. Fazer Reserva\n");

     printf("8. Cancelar Reserva\n");

     printf("9. Menu de Pesquisa\n");

     printf("10. Consultar Fidelidade\n");

     printf("11. Sair\n");

     printf("Escolha uma opção: ");



     scanf("%d", &opcao);

     limparBufferMain();
```

```
    switch (opcao) {

        case 1:

            cadastrarPassageiro();

            break;



        case 2:

            listarPassageiros();

            break;



        case 3:

            cadastrarTripulacao();

            break;



        case 4:

            listarTripulacao();

            break;



        case 5:

            cadastrarVoo();

            break;



        case 6:

            listarVoos();

            break;
```

```c
        case 7:
            cadastrarReserva();
            break;

        case 8:
            baixarReserva();
            break;

        case 9:
            menuPesquisa();
            break;

        case 10: {
            int codigo;
            printf("Digite o código do passageiro: ");
            scanf("%d", &codigo);
            limparBufferMain();
            consultarPontosFidelidade(codigo);
            break;
        }

        case 11:
            printf("\nEncerrando o sistema...\n");
```

```
            return 0;



        default:

            printf("Opção inválida! Escolha um número entre 1 e 11.\n");

    }

}



return 0;

}
```