Documentação do Sistema de Gerenciamento de Companhia Aérea

Carlos Eduardo S. Santos, Kayke Emanoel de Souza, Rafael Rocha C. Brant

13 de dezembro de 2024

Introdução

Trabalho Prático: AEDS 1 e Fund. de Eng. de Software

Este projeto faz parte dos trabalhos práticos das disciplinas de Algoritmos e Estruturas de Dados 1 e Fundamentos de Engenharia de Software do curso de Engenharia de Software na PUC Minas. O objetivo é aplicar conceitos teóricos na prática, incluindo modelagem de sistemas, uso de algoritmos eficientes e implementação de boas práticas de programação orientada a objetos.

1 Repositório do Projeto

O código-fonte completo e as instruções detalhadas para compilar e executar o sistema estão disponíveis no repositório GitHub: `https://github.com/Eman134/Trabalho-CompanhiaAerea`. Incluímos um arquivo README com requisitos de sistema e exemplos de execução.

2 Estrutura do Projeto

O projeto segue uma organização modular para facilitar a manutenção e expansão. Os diretórios e arquivos principais incluem:

- `main.cpp`: Arquivo principal que orquestra o fluxo do programa.
- `controllers/`: Diretório contendo os controladores do sistema, que implementam lógica de negócios:
  - `FazerReserva.h`: Responsável por funcionalidades de reserva de assentos em voos. Faz uso do controlador `VooController`.
  - `Passageiro.h`: Define a classe modelo para representar passageiros, com atributos como `codigo_passageiro` e `nome`.
  - `AviaoController.h`: Gerencia as operações relacionadas a aviões, com uma lista de aviões (`vector<Aviao>`).
  - `TripulacaoController.h`: Lida com a equipe de tripulação, armazenando informações em uma lista de tripulações (`vector<Tripulacao>`).

  - `VooController.h`: Controla os voos, integrando funcionalidades com aviões e tripulações.

- `models/`: Diretório com as classes modelo que representam as entidades do sistema:
  - `Assento.h`: Representa assentos no avião, conectando-se a passageiros.
  - `Aviao.h`: Define o modelo de avião, com atributos como `codigo`, `nome` e `quantidade de assentos`.
  - `Tripulacao.h`: Representa membros da equipe, incluindo `codigo`, `nome` e `cargo`.
  - `Voo.h`: Modelo para informações de um voo, com atributos como `codigo`, lista de assentos e outros detalhes relevantes.

3 Funções e Parâmetros Utilizados no Programa

As funções descritas nas classes `AviaoController`, `PassageiroController`, `TripulacaoControll` `VooController`, e `ReservaController` são responsáveis pela manipulação dos dados associados a seus respectivos modelos. Elas servem como intermediárias entre as camadas de dados (modelos) e a interface do usuário ou outras partes do sistema que requerem acesso a esses dados.

A seguir, as principais funções de cada controlador e sua interação com os modelos são descritas:

3.1 Classe `AviaoController`

- `AviaoController()`
- `void carregarAvioes()`
- `void salvarAvioes()`
- `Aviao* buscarAviao(int codigo_aviao)`
- `vector<Aviao> getListaAvioes() const`
- `void cadastrarAviao()`
- `void editarAviao(int codigo_aviao)`
- `void excluirAviao(int codigo_aviao)`
- `void visualizarAvioes() const`
- `void setDisponibilidade(int codigo_aviao, bool disponibilidade)`
- `int avioesDisponiveis() const`
- `int avioesCadastrados() const`
- `int getProximoCodigo() const`

3.2 Classe PassageiroController

- PassageiroController()
- void carregarPassageiros()
- void salvarPassageiros()
- Passageiro* buscarPassageiro(int codigo_passageiro)
- vector<Passageiro> getListaPassageiros() const
- void cadastrarPassageiro()
- void editarPassageiro(int codigo_passageiro)
- void excluirPassageiro(int codigo_passageiro)
- void visualizarPassageiros() const
- int getProximoCodigo() const

3.3 Classe TripulacaoController

- TripulacaoController()
- vector<Tripulacao> getTripulacoes() const
- int getQtdPiloto() const
- int getQtdCopiloto() const
- int getQtdComissario() const
- void cadastrarTripulacao()
- void listarTripulacao()
- void salvarTripulacao()
- void carregarTripulacao()

3.4 Classe VooController

- VooController()
- void carregarVoos()
- Voo* buscarVoo(int codigo_voo)
- int getNumeroVoos() const
- vector<Voo> getListaVoos() const
- void salvarVoos()
- void cadastrarVoo(AviaoController* aviaoController, TripulacaoController* tripulacaoController)
- void visualizarVoos() const
- void darBaixaVoo(int codigo_voo)
- int voosCadastrados() const
- int getProximoCodigo() const

3.5 Classe ReservaController

- ReservaController()
- void cadastrarReserva(VooController* vooController, PassageiroController* passageiroController)
- void exibirDetalhesReserva(Reserva* reserva)
- Reserva* buscarReserva(int codigo_reserva)
- vector<Reserva> getListaReservas() const
- void salvarReservas()
- void carregarReservas()
- void excluirReservasVoo(int codigo_voo)
- void listarReservas()
- void exibirTabelaAssentos(int codigo_voo)
- void reservarAssento(int codigo_voo, int assento, Passageiro* passageiro)
- int getProximoCodigoReserva() const
- bool estaReservado(int codigo_voo, int assento)

3.6 Modelos de Dados

Os modelos principais utilizados para manipulação dos dados incluem:

3.6.1 Aviao

- Aviao(int codigo_aviao, string nome_aviao, int qtd_assentos, bool disponivel)
- int getCodigoAviao() const
- string getNomeAviao() const
- int getQtdAssentos() const
- bool getDisponivel() const
- void setDisponibilidade(bool disponibilidade)
- ...

3.6.2 Passageiro

- Passageiro(int codigo_passageiro, string nome, string endereco, string telefone, bool fidelidade, int pontos_fidelidade)
- int getCodigoPassageiro() const
- string getNome() const
- ...

3.6.3 Tripulacao

- `Tripulacao(int codigo, string nome, string cargo, string telefone, bool disponivel)`
- `int getCodigoTripulacao() const`
- `string getNomeTripulacao() const`
- ...

3.6.4 Voo

- `Voo()`
- `int getCodigoVoo() const`
- `string getData() const`
- ...

3.6.5 Reserva

- `Reserva()`
- `int getCodigoReserva() const`
- `string getNomePassageiro() const`
- ...

4 Instruções de Compilação, Execução e Uso

- `Como Executar`: Para executar o projeto, basta clonar o repositório e executar o arquivo TrabalhoCompanhiaAerea.exe, localizado na pasta principal do projeto. Se preferir, você pode compilar o projeto com g++ ou outro compilador de sua preferência.

- `Como Compilar`: Pré-requisitos: g++ instalado ou qualquer outro compilador C++. No Windows, é recomendado o uso do MinGW ou WSL (Windows Subsystem for Linux) para compilar. No Linux/macOS, o g++ pode ser instalado facilmente via gerenciador de pacotes. Passos para compilar: Clone o repositório, Compile o projeto com o seguinte comando:

  g++ main.cpp src/*/*.cpp -o TrabalhoCompanhiaAerea

  Após compilar, execute o programa gerado (no Windows, será um arquivo TrabalhoCompanhiaAerea.exe): ./TrabalhoCompanhiaAerea

- `Layout`: O layout do sistema é textual, utilizando menus simples de navegação no console. O usuário interage por meio da digitação de números para selecionar as opções desejadas, e o programa exibe informações no formato de texto para as operações realizadas.

5 Documentação Inline no Código

Toda a documentação detalhada sobre o funcionamento do sistema está presente como comentários dentro dos arquivos do código-fonte. Os comentários seguem as melhores práticas de desenvolvimento e estão estrategicamente colocados para facilitar a compreensão:

- Declarações de Classes e Métodos: Cada classe e método possui comentários explicando sua finalidade, entradas e saídas.
- Projects no GitHub: Além da documentação inline, a aba **Projects** no repositório GitHub do projeto contém um *backlog* e um *roadmap* detalhados. Ambos estão disponíveis para consulta e atualização colaborativa.

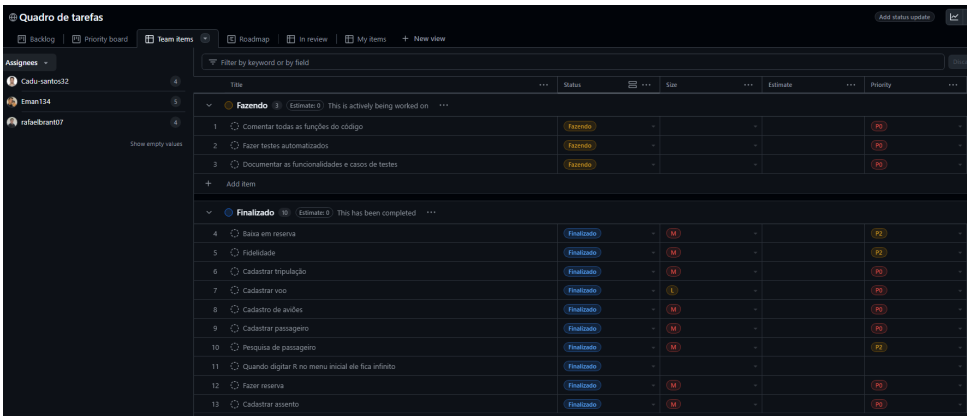

Figure 1: Quadro dos itens do projeto

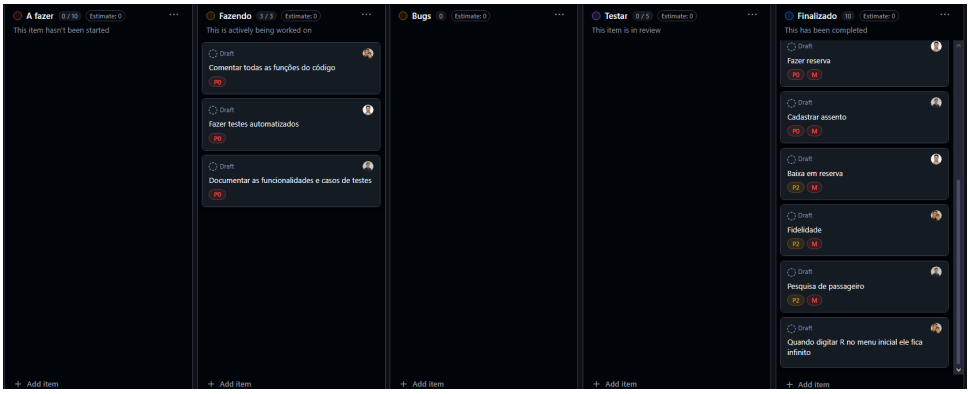

Figure 2: Quadro do KANBAN backlog

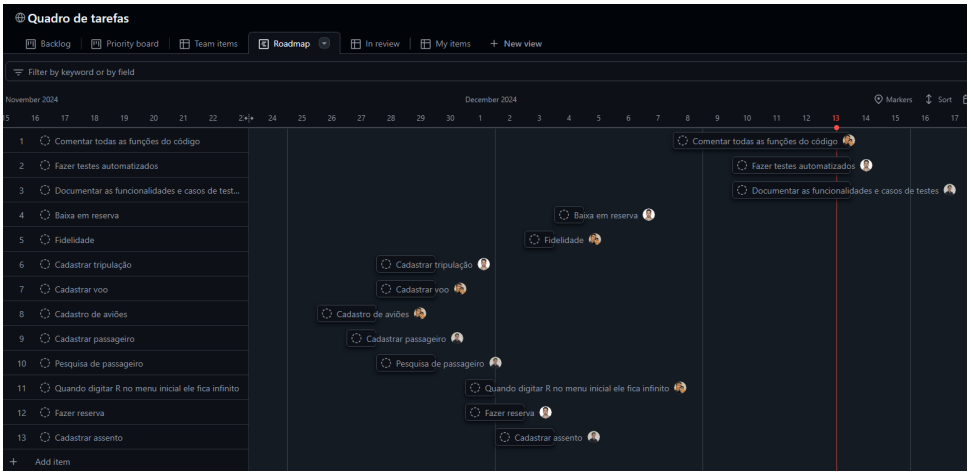

Figure 3: ROADMAP das tarefas

Caso de teste 1:	Resumo: Garantir que não haja dois passageiros com mesmo codigo
Prioridade: Alta	status: Aprovado

Pré-condição: Ter cadastrado ao menos um passageiro, para garantir que os passageiros estão tendo codigos diferente

Passos:
1 - Acessar o sistema
2 - digitar a tecla "5"
3 - preencher nome
4 - preencher endereço

Resultado esperado: "Passageiro cadastrado com sucesso."
Observações: os codigos são cadastrado automaticamente, somando com o ultimo.
comentários: sem comentário.

Figure 4: CASO DE TESTE 1

Caso de teste 2:	Resumo: cada membro da tripulacao deve ter um cargo específico.
Prioridade: Media	status: Aprovado

Pré-condição: Não existe pré-condição.

Passos:
1 - Acessar o sistema
2 - digitar a tecla "3"
3 - preencher nome
4 - preencher telefone
5 - digitar o código do cargo

Resultado esperado: "Tripulação cadastrada com sucesso."
Observações: os codigos são cadastrado automaticamente, somando com o ultimo.
comentários: sem comentário, e os cargos são escolhidos por número.

Figure 5: CASO DE TESTE 2

Caso de teste 3:	Resumo: deve garantir que não haja dois tripulantes com o mesmo codigo.
Prioridade: Media	status: Aprovado

Pré-condição: Já ter criado pelo menos um tripulante.

Passos:
1 - Acessar o sistema
2 - digitar a tecla "3"
3 - preencher nome
4 - preencher telefone
5 - digitar o código do cargo

Resultado esperado: "Tripulação cadastrada com sucesso."
Observações: os codigos são cadastrado automaticamente somando com o ultimo.

Figure 6: CASO DE TESTE 3

Caso de teste 4:	Resumo: no voo pode cadastrar informações sobre data, hora, origem, destino, tarifa, tripulação e o avião.
Prioridade: Alta	status: Aprovado

Pré-condição: Já ter criado pelo menos um tripulante, e um avião.

Passos:
1 - Acessar o sistema
2 - digitar a tecla "7"
3 - preencher data
4 - preencher hora
5 - preencher origem
6 - preencher destino
7 - preencher tarifa
8 - digitar o codigo do tripulante e do avião
Resultado esperado: "Voo cadastrado com sucesso."
Observações: sem obserações..

Figure 7: CASO DE TESTE 4

Caso de teste 5:	Resumo: deve verificar a presença de ao menos um piloto e um copiloto para que o voo seja marcado como ativo.
Prioridade: Alta	status: Aprovado

Pré-condição: Já ter criado pelo menos um piloto e um co-piloto, e um avião.

Passos:
1 - Acessar o sistema
2 - digitar a tecla "10"
3 - preencher código do voo
4 - preencher código passageiro
5 - escolher os assentos que estão disponíveis
Resultado esperado: "código de reserva, código do voo, código de passageiro, e chama a função de exibirdetalhespassageiro()."
Observações: A reserva é feita em outra classe chamada Reserva.

Figure 8: CASO DE TESTE 5

Caso de teste 6:	Resumo: deve ser possível cadastrar os assentos em cada voo.
Prioridade: Alta	status: Aprovado
Pré-condição: Já ter criado pelo menos um piloto e um co-piloto, e um avião.	
Passos: 1 - Acessar o sistema 2 - digitar a tecla “7" 3 - digita o codigo do avião que deseja associar ao voo	
Resultado esperado: “voo cadastrado com sucesso” Observações: os assentos são passados quando cria um avião.	

Figure 9: CASO DE TESTE 6