PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE
Bacharelado em Engenharia de Software

Nome dos integrantes do grupo: Davi Mendes, Paulo Tadeu Pereira
Nome do sistema: Gerenciamento de Voos

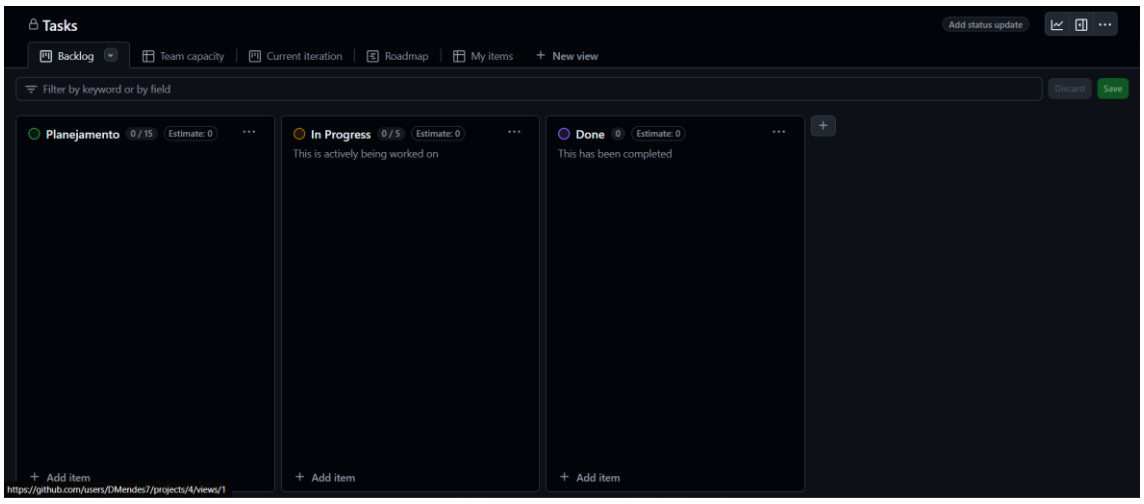
Apresentação:

O Sistema de Gerenciamento de Voos é um software desenvolvido para a companhia aérea Voo Seguro. Ele organiza e gerencia as operações relacionadas a passageiros, tripulação, voos, assentos e reservas, resolvendo problemas como reservas duplicadas e falta de controle sobre disponibilidade de assentos.

Backlog do produto:

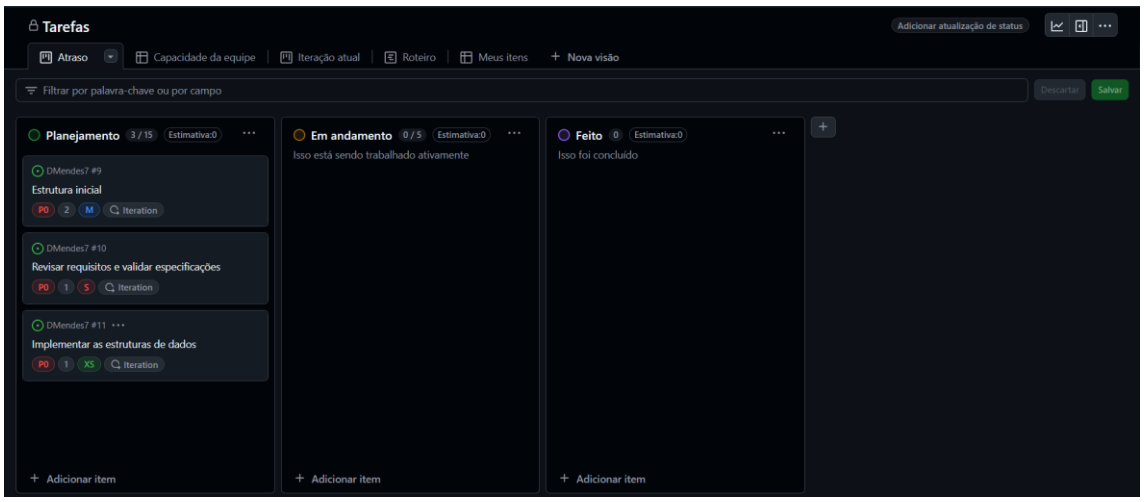
A Figura 1 apresenta o time criado no GitHub Projects com as divisões das tarefas. O quadro principal do projeto foi configurado para organizar as funções entre os integrantes e quais tarefas deveriam ser entregues por cada integrante em ordem de prioridade e sprints. Cada coluna no quadro representa uma etapa, como "Backlog", "Em Progresso" e "Concluído", facilitando a visualização do progresso. Os cartões detalham as tarefas atribuídas a cada integrante, incluindo descrições, checklists e prazos definidos.

Figura 1 – Time e quadros criados no Github



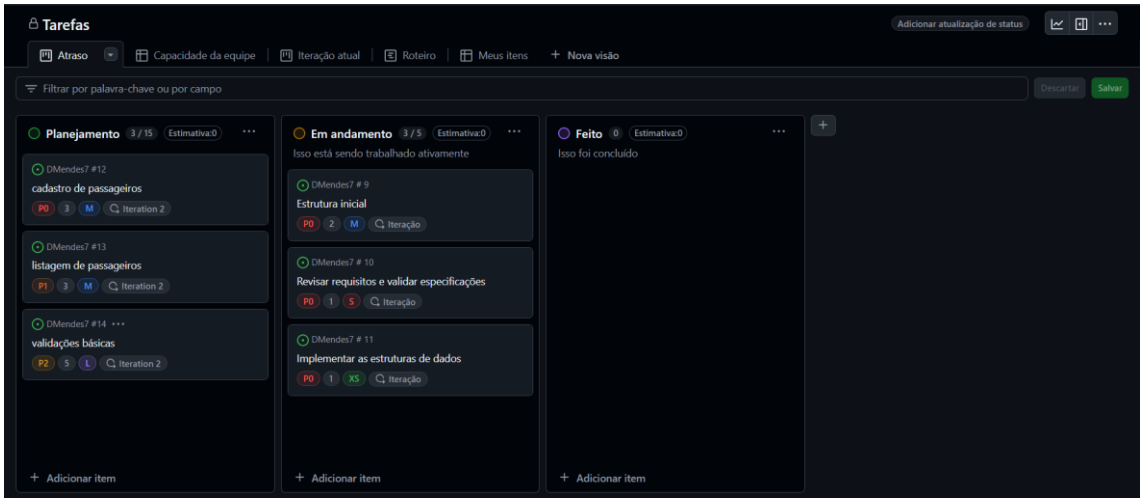
A Figura 2 apresenta o protótipo de tela realizado no início do projeto mostrando no quadro principal do **GitHub Projects** a divisão das funções e quais tarefas os integrantes deveriam finalizar em cada sprint. Cada tarefa foi organizada em cartões e categorizada por prioridade e etapa de desenvolvimento. Os itens marcados como "Alta Prioridade" foram atribuídos à primeira sprint, enquanto as tarefas subsequentes foram organizadas conforme sua relevância para as entregas das sprints seguintes.

Figura 2 - Backlog geral



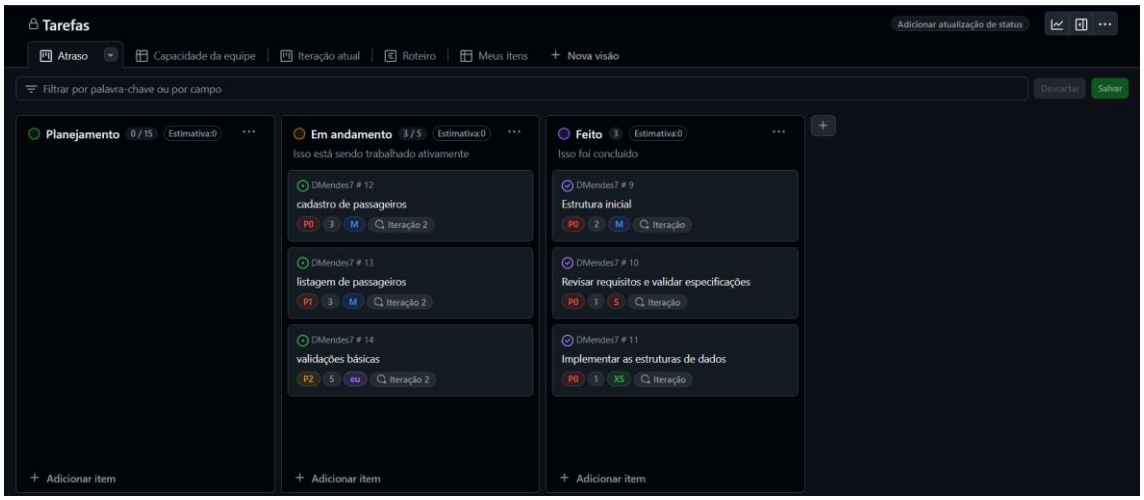
A Figura 3 demonstra o status do backlog do produto no andamento da primeira sprint e o planejamento da segunda sprint.

Figura 3 – Andamento do fluxo do Backlog



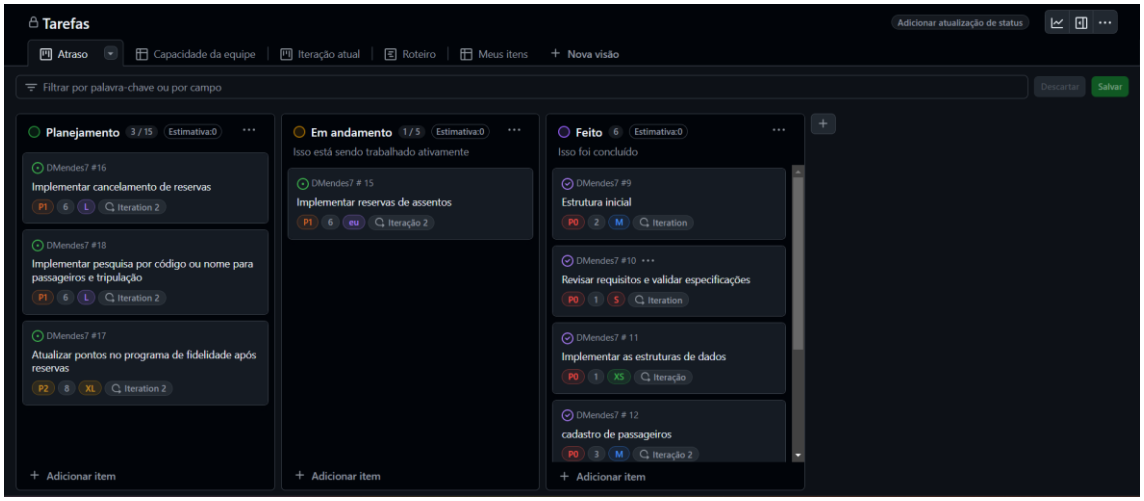
A próxima figura ilustra o progresso do projeto, destacando as transições realizadas entre as diferentes sprints. A imagem representa as tarefas finalizadas, as pendências reavaliadas e as etapas superadas ao longo do desenvolvimento. Essas transições evidenciam tanto o avanço das funcionalidades inicialmente planejadas quanto os ajustes implementados, assegurando a evolução constante do projeto.

Figura 4 – Finalização e início entre as Sprints



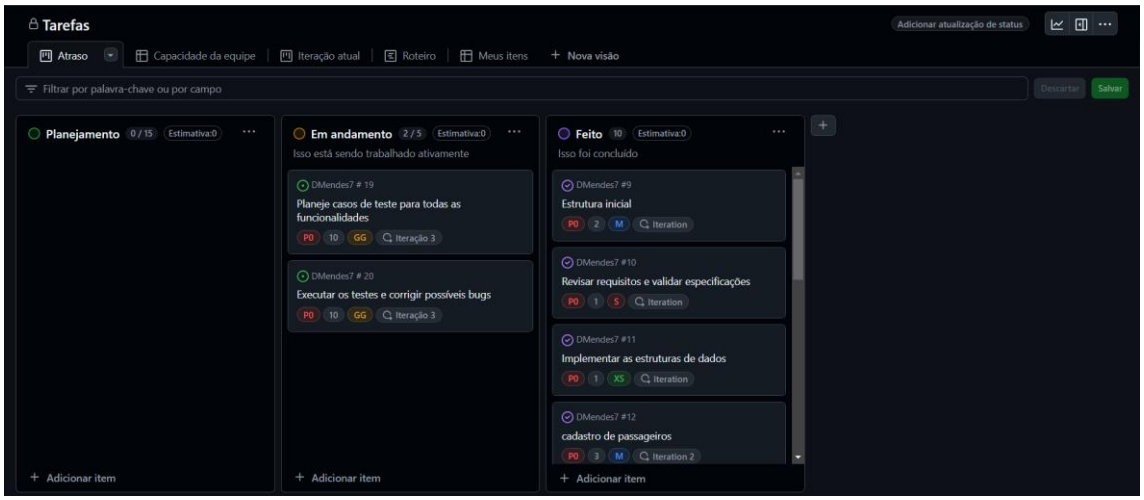
A imagem a seguir ilustra as funcionalidades avançadas incorporadas ao planejamento. Essas funcionalidades foram planejadas para expandir as capacidades do projeto, integrando recursos mais sofisticados e ajustados a requisitos específicos, com o objetivo de aprimorar a eficiência geral do sistema. Além disso, foram concebidas para proporcionar uma experiência mais abrangente, atendendo às demandas identificadas nas etapas iniciais do desenvolvimento.

Figura 5 –Início das Funcionalidades Avançadas



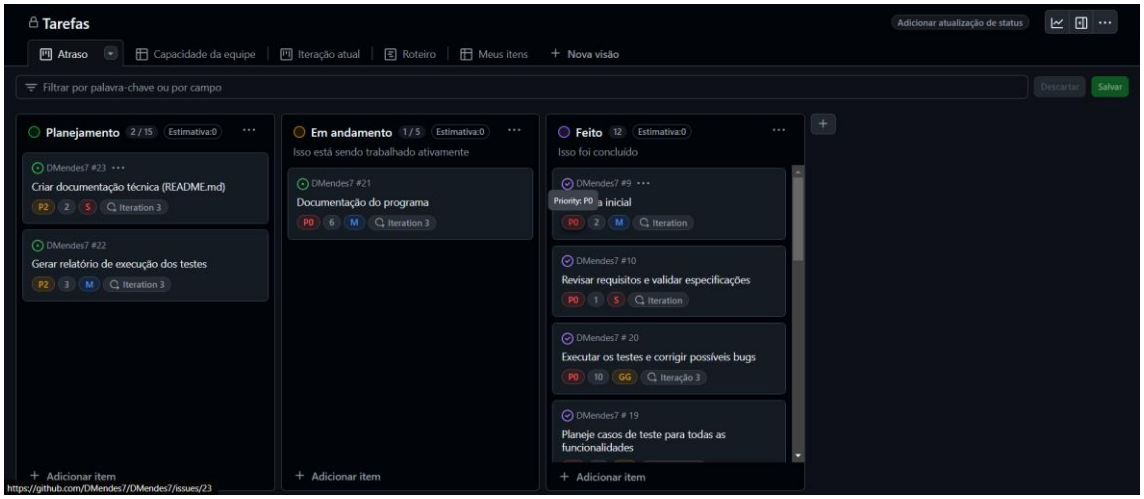
A imagem a seguir representa os testes e validações em fase de execução no sistema. Essas etapas foram fundamentais para garantir a qualidade e o funcionamento adequado das funcionalidades implementadas. Os testes envolveram a verificação dos critérios de aceitação definidos no planejamento, além de validações técnicas para assegurar a robustez e a confiabilidade do sistema.

Figura 6 –Início dos Testes e Validações



A imagem a seguir representa a documentação em andamento no projeto. Essa etapa foi essencial para registrar o funcionamento do sistema, detalhar as funcionalidades implementadas, e orientar futuros desenvolvedores ou usuários. A documentação inclui diagramas, descrição de processos, assinaturas de funções e parâmetros, além de exemplos de uso e casos de teste.

Figura 7 –Planejamento e execução da Documentação



A imagem a seguir representa a etapa de revisão e entrega do projeto. Nessa fase, todas as funcionalidades foram avaliadas, ajustes finais foram realizados e o sistema foi preparado para sua entrega oficial. Essa etapa incluiu a validação dos requisitos, a revisão da documentação e a verificação final de qualidade, garantindo que o produto atendesse às expectativas e estivesse pronto para uso.

Figura 8 –Fase final - Revisão

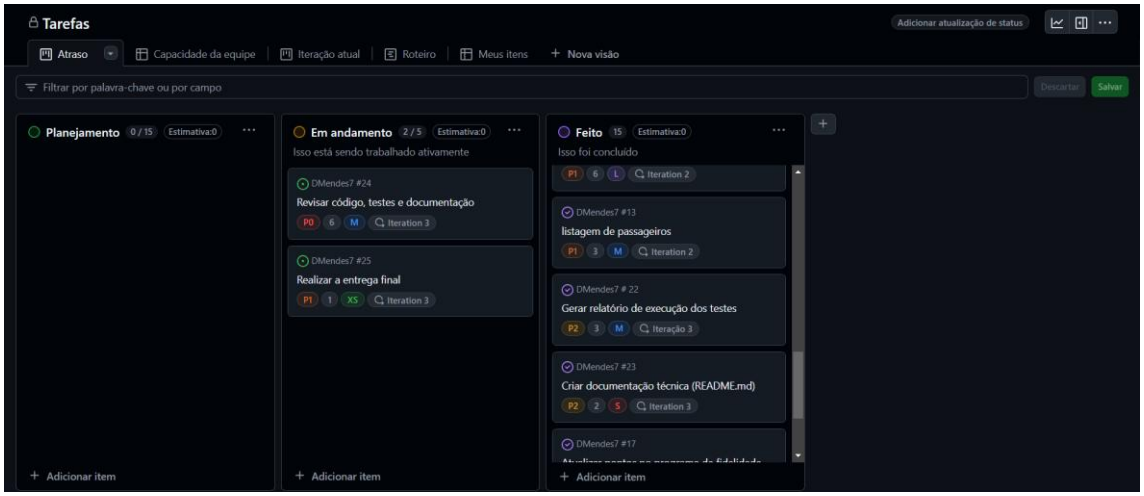
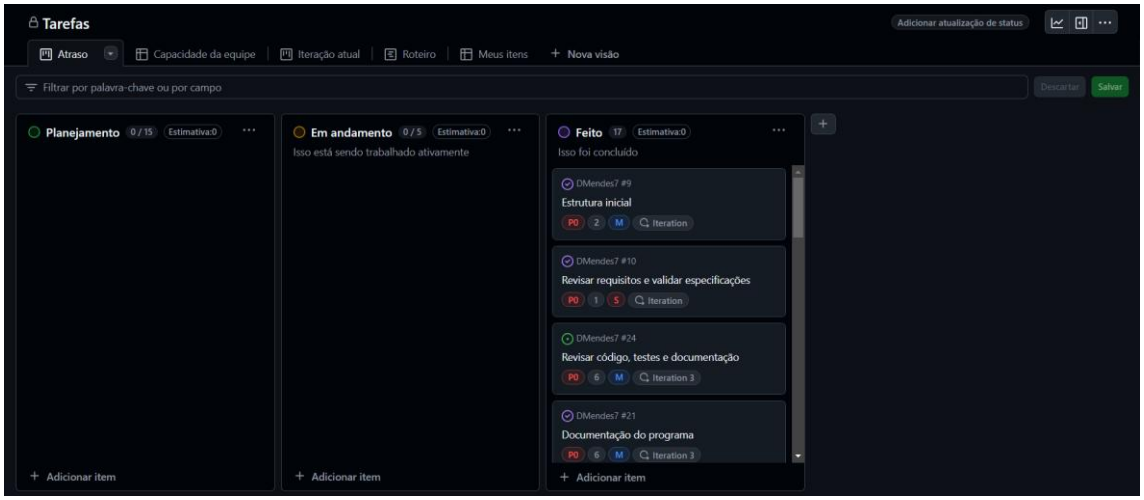


Figura 9 –Fase final – Entrega



Lista de Assinaturas das Funções e Parâmetros

Explicação da estrutura de dados principal do programa

O programa utiliza estruturas de dados para representar passageiros, tripulação, voos, assentos e reservas. Os dados são manipulados por meio de funções que interagem com arquivos para garantir persistência.

Apresentação das Assinaturas das Funções

1. int gerarCodigo(const char arquivo)

Função para gerar códigos únicos.
Recebe como parâmetro o nome do arquivo (arquivo) onde estão registrados os dados. A função analisa o maior código existente no arquivo e retorna um código único para novos registros.

- **Parâmetros:**
  - const char *arquivo: Nome do arquivo a ser analisado.
- **Retorno:**
  - O próximo código disponível como inteiro.

---

2. void menu()

Função para exibir o menu principal.
Exibe as opções de funcionalidades disponíveis para o usuário.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

3. void cadastrarPassageiro()

Função para cadastrar um passageiro.
Recebe os dados do usuário (nome, endereço, telefone, fidelidade) e salva o registro no arquivo passageiros.txt.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

4. void listarPassageiros()

Função para listar os passageiros cadastrados.
Lê os registros do arquivo passageiros.txt e exibe as informações no console.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

5. void cadastrarTripulacao()

Função para cadastrar um membro da tripulação.
Solicita dados como nome, telefone e cargo (Piloto, Copiloto ou Comissário), salvando no arquivo tripulacao.txt.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

6. void listarTripulacao()

Função para listar os membros da tripulação cadastrados.
Lê os registros do arquivo tripulacao.txt e exibe os dados no console.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

7. void cadastrarVoo()

Função para cadastrar um voo.
Solicita informações como data, hora, destino, piloto, copiloto e tarifa. Verifica a presença de piloto e copiloto para ativar o voo, e salva o registro no arquivo voos.txt.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

8. void listarVoos()

Função para listar os voos cadastrados.
Lê os registros do arquivo voos.txt e exibe os dados no console.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

9. void cadastrarAssento()

Função para cadastrar assentos para um voo.
Associa números de assento a um voo específico e salva os dados no arquivo assentos.txt.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

10. void listarAssentos()

Função para listar os assentos de um voo.
Lê o arquivo assentos.txt e exibe os números de assento e seus status (livre ou ocupado).

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

11. void realizarReserva()

Função para realizar uma reserva de assento.
Recebe o código do voo, número do assento e código do passageiro. Atualiza o status do assento para ocupado e registra a reserva no arquivo reservas.txt.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

12. void baixaReserva()

Função para cancelar uma reserva.
Recebe o código do voo e número do assento, atualiza o status do assento para livre e remove a reserva associada.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

13. void programaFidelidade()

Função para listar pontos de fidelidade dos passageiros.
Exibe os pontos acumulados de passageiros fidelizados, calculados com base nas reservas realizadas.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

---

14. void pesquisar()

Função para pesquisar passageiros ou tripulação.
Permite buscar registros por código ou nome. Exibe os resultados encontrados.

- **Parâmetros:** Nenhum.
- **Retorno:** Nenhum.

Caso de Teste: Realizar Reserva

Entradas	**Classes Válidas**	**Resultado Esperado**	**Classes Inválidas**	**Resultado Esperado**
Código do voo (int), Número do assento (int), Código do passageiro (int).	Código do voo existente, número do assento livre, código de passageiro válido. Exemplo: Voo 1001, Assento 5, Passageiro 10.	Reserva realizada com sucesso. O status do assento é alterado para "ocupado" e o registro é salvo no arquivo reservas.txt.	Código do voo inexistente, número do assento inválido (não cadastrado ou já ocupado), código de passageiro inexistente. Exemplo: Voo 2000, Assento 20, Passageiro 99.	Reserva não realizada. O sistema exibe uma mensagem de erro indicando o problema específico e solicita uma nova entrada.