CENTRO UNIVERSITÁRIO FEI
Guilherme Proença de Souza
Vitor da Silva Rodrigues de Souza

PROJETO FEIFOOD

Relatório Final do Projeto apresentado ao Centro Universitário FEI, como parte dos requisitos da Disciplina “CCP230

São Bernado do Campo
2025

SUMÁRIO

1 INTRODUÇÃO

O FEIFood é uma aplicação de linha de comando que simula uma plataforma de pedidos de comida inspirada em serviços do mercado. O objetivo do projeto foi implementar funcionalidades para usuários e administradores, com persistência simples em arquivos de texto.

Tecnologias utilizadas: linguagem C; persistência por arquivos texto (.txt); organização modular em múltiplos arquivos-fonte e headers.

Escopo: implementação de cadastro e login (usuário e administrador), listagem e gerenciamento de alimentos e estabelecimentos, criação/edição/exclusão de pedidos, avaliação de pedidos e consulta de estatísticas básicas.

2 DESENVOLVIMENTO

Estrutura dos arquivos de dados

- Arquivos principais:
    - **usuarios.txt** — cada linha: nome|senha
    - **administradores.txt** — cada linha: nome|senha
    - **estabelecimentos.txt** — cada linha: id|nome
    - **alimentos.txt** — cada linha: id|nome|idEstab
    - **pedidos.txt** — cada linha: id|nomeUsuario|listaAlimentos|avaliacao
- Formato: campos separados por pipe (|). Quando necessário atualizar registros, o padrão usado é reescrever o arquivo por meio de um arquivo temporário dados/temp.txt, removendo e renomeando para garantir consistência.

Módulos e principais funções

Arquivo main (fluxo e menus)

- Exibe menu principal com opções: cadastrar usuário, login usuário, login administrador, cadastrar administrador e sair.
- Após login válido, direciona para:
    - **menuUsuario(nome)** — opções: fazer pedido, excluir pedido, avaliar pedido, adicionar alimento ao pedido.
    - **menuAdmin()** — opções: cadastrar/excluir alimento, cadastrar estabelecimento, consultar usuários, mostrar estatísticas.

usuario.c

- **cadastrarUsuario()**: pede nome e senha e grava em dados/usuarios.txt no formato nome|senha.
- *loginUsuario(char* nome, char senha)**: abre dados/usuarios.txt e compara nome e senha lidos linha a linha; retorna 1 em caso de sucesso e 0 caso contrário.

Observações: autenticação simples por comparação em texto plano.

admin.c

- **cadastrarAdministrador()**: lê nome e senha, salva em dados/administradores.txt como nome|senha.
- *loginAdministrador(char* nome, char senha)**: valida via leitura de administradores.txt.
- **cadastrarEstabelecimento()**: gera ID aleatório, lê nome do estabelecimento (com espaços) e grava id|nome em estabelecimentos.txt.
- **consultarUsuarios()**: lê usuarios.txt e imprime a lista de nomes.
- **mostrarEstatisticas()**: conta linhas de pedidos.txt para apresentar o total de pedidos.

Observações: geração de IDs com rand(); possibilidade de colisão de IDs (probabilidade baixa, mas presente). Mensagens de erro tratam abertura de arquivos via perror.

alimentos.c

- **cadastrarAlimento()**:
    - Gera ID aleatório.
    - Lê nome do alimento (suporta espaços).
    - Lista estabelecimentos lendo estabelecimentos.txt e mostra ao usuário.
    - Lê ID do estabelecimento escolhido e grava idAlimento|nomeAlimento|idEstab em alimentos.txt.
- **listarAlimentos()**:
    - Carrega estabelecimentos para memória (vetor Estab) e, em seguida, percorre alimentos.txt imprimindo: id - nome (nomeEstabelecimento).
- **alimentoExiste(int id)**: percorre alimentos.txt verificando existência pelo ID.
- **excluirAlimento()**: solicita ID, valida existência e reescreve alimentos.txt sem o alimento removido usando temp.txt.

Observações: leitura de estabelecimento feita antes do cadastro de alimento para permitir associação e exibição clara ao usuário.

pedido.c

- *criarPedido(char* nomeUsuario, int idAlimento)*:
    - Gera ID aleatório, grava em pedidos.txt no formato id|nomeUsuario|idAlimento|0 (avaliacao inicia em 0) e retorna o ID.
- *fazerPedido(char* nomeUsuario)*:
    - Lista alimentos, solicita ID do alimento, valida existência e chama criarPedido.
- **pedidoExiste(int idBuscado)**: verifica a presença do ID em pedidos.txt.
- **excluirPedido(int idPedido)**: valida e reescreve pedidos.txt sem o pedido removido.
- **avaliarPedido(int idPedido, int estrelas)**:
    - Valida existência, reescreve o arquivo substituindo a avaliação do pedido por estrelas.
- **adicionarAlimentoAoPedido(int idPedido, int idAlimento)**:
    - Valida existência do pedido, lê cada linha e, para o pedido alvo, concatena o novo ID de alimento à lista existente (separada por vírgula) e reescreve o arquivo.

Observações: lista de alimentos dentro de um pedido armazenada como uma string com IDs separados por vírgula; operações de edição usam cópia via temp.txt. A avaliação sobrescreve o campo avaliação do pedido.

utils.c

- **limparBuffer()**: consome caracteres até newline, usado por pausarTela.
- **pausarTela()**: pede ENTER ao usuário (simples controle de fluxo).
- **gerarIdUnico()**: gera ID aleatório via srand(time(NULL)); retorna rand() % 10000.

Observações: srand é chamada dentro de gerarIdUnico — em outras partes do código rand() também é usado diretamente; ideal chamar srand apenas uma vez no início para evitar repetição de seed em chamadas múltiplas próximas no tempo.

3 RESULTADOS

- nterface: menus de texto funcionais para usuário e administrador; navegação por números simples.
- Persistência: todos os cadastros e alterações são persistidos nos arquivos dentro de dados/. Testes realizados:

    - Cadastro de usuário adiciona nova linha em usuarios.txt.
    - Cadastro de estabelecimento e alimento gera IDs e associações corretamente em estabelecimentos.txt e alimentos.txt.
    - Criação de pedido registra id|nomeUsuario|idAlimento|0 em pedidos.txt.
    - Adição de alimento a pedido atualiza a lista de alimentos para incluir o novo ID separado por vírgula.
    - Avaliação altera o campo de avaliação no registro do pedido.
    - Exclusão de alimento/pedido reescreve os arquivos removendo a entrada alvo.
- Validações implementadas:
    - Verificação de existência de alimento antes de adicionar a pedido.
    - Verificação de existência de pedido antes de avaliá-lo ou excluí-lo.
    - Mensagens informativas em casos de erro (arquivo não encontrado, ID inexistente, login inválido).

Limitações observadas:

- Autenticação em texto plano (sem hashing).
- Possível colisão de IDs gerados randômicamente.
- Concorrência não tratada (acesso simultâneo a arquivos pode causar inconsistências).
- Formato de armazenagem simples impede consultas complexas sem parsing adicional.

4 CONCLUSÕES

O FEIFood implementa um conjunto coerente de funcionalidades de um sistema de pedidos usando C e arquivos texto, atendendo aos requisitos propostos para a disciplina. A modularização facilitou o desenvolvimento e a manutenção: cada domínio (usuário, administrador, alimento, pedido, utilitários) possui responsabilidades bem definidas.

Para evolução futura, recomenda-se:

- Unificação da geração de IDs em uma rotina única com verificação de colisão.
- Uso de hashing para senhas.

- Migração para um formato de armazenamento estruturado (por exemplo CSV com bibliotecas ou SQLite) para permitir consultas mais robustas e evitar regravações inteiras de arquivos.
- Tratamento de entrada mais robusto (evitar scanf direto quando possível; validar limites).
- Inicializar a seed do gerador aleatório uma vez no início do programa.

5 REFERÊNCIAS

Kernighan B., Ritchie D. The C Programming Language.
Documentação de funções da biblioteca padrão C (stdio.h, stdlib.h, string.h).