Curso Ebac - Ti do Zero

Registro de Nomes
Documento de Arquitetura de Software

Versão <1.0>

Índice Analítico

Documento de Arquitetura de Software

1. Introdução

O presente documento descreve a arquitetura do **Sistema de Gerenciamento de Usuários do Escritório EBAC**, um projeto desenvolvido para fins acadêmicos. O sistema tem como objetivo facilitar o cadastro, consulta e exclusão de usuários dentro de um ambiente corporativo simulado.

A aplicação permite que os administradores realizem o **registro de novos usuários**, inserindo informações como CPF, nome, sobrenome e cargo. Além disso, oferece a funcionalidade de **consulta de usuários cadastrados**, permitindo a análise dos dados registrados. Por fim, o sistema possibilita a **exclusão de usuários**, removendo completamente suas informações do banco de dados.

O sistema foi projetado para ser simples e eficiente, garantindo a integridade e confidencialidade dos dados armazenados.

1.1 Finalidade

O **Sistema de Gerenciamento de Usuários do Escritório EBAC** foi desenvolvido com o propósito de adquirir, armazenar e gerenciar informações de usuários dentro de um ambiente corporativo fictício. Sua principal funcionalidade é garantir um controle eficiente sobre os dados cadastrados, permitindo que sejam consultados e analisados conforme necessário, além de possibilitar sua exclusão quando desejado.

O sistema oferece três operações essenciais:

Registro de Usuários: Captura e armazena informações como CPF, nome, sobrenome e cargo, garantindo que os dados fiquem disponíveis para futuras consultas.
Consulta de Usuários: Permite a visualização e análise das informações cadastradas, possibilitando a obtenção de insights sobre os dados armazenados.
Exclusão de Usuários: Oferece a opção de remover permanentemente os dados de um usuário do sistema, garantindo o controle sobre a retenção das informações.

Com essas funcionalidades, o sistema proporciona um ambiente estruturado para o gerenciamento de dados, assegurando organização e acessibilidade das informações de forma simples e intuitiva.

1.2 Escopo

O **Sistema de Gerenciamento de Usuários do Escritório EBAC** foi projetado para fornecer um ambiente centralizado e eficiente para o gerenciamento de dados de usuários dentro de um escritório de forma simples, segura e acessível.

1.3 Definições, Acrônimos e Abreviações

O Projeto conta com uma série de definições como:

Registro: Captura e armazena informações como CPF, nome, sobrenome e cargo, garantindo que os dados fiquem disponíveis para futuras consultas ordenadas em nosso banco de dados.

Consulta: Responsável por acessar de forma intrusiva o nosso banco de dados e a partir de uma chave primaria apontada ao CPF do usuário a partir dessa chave primaria nosso código acessa esse banco de dados e retorna nossas informações antes cadastradas.

Exclusão: Utiliza do mesmo pressuposto de busca do sistema de consultas, mas com o intuito de exclusão de informações antes registradas e armazenadas em nosso banco de dados

1.4 Visão Geral

Este documento tem como objetivo detalhar a arquitetura do **Sistema de Gerenciamento de Usuários do Escritório EBAC**, descrevendo suas camadas, componentes, fluxos e requisitos técnicos. Ele está estruturado para fornecer uma visão abrangente do sistema, permitindo que desenvolvedores, arquitetos e demais envolvidos compreendam sua organização e funcionamento.

O documento está organizado da seguinte forma:

- **Representação Arquitetural**: Apresenta a estrutura do sistema, abordando as principais visões da arquitetura, como casos de uso, lógica do sistema, processos e implantação.
- **Metas e Restrições da Arquitetura**: Define requisitos técnicos e restrições que impactam a arquitetura, como segurança, privacidade e tecnologias utilizadas.
- **Visão de Casos de Uso**: Lista os principais casos de uso do sistema, destacando sua importância para a arquitetura.
- **Realizações de Casos de Uso**: Ilustra a implementação de casos de uso selecionados, demonstrando como os componentes do sistema interagem para atender às funcionalidades.
- **Visão Lógica**: Detalha a estrutura modular do sistema, apresentando seus subsistemas, pacotes e classes mais relevantes.
- **Visão de Processos**: Explica a organização do sistema em processos e threads, bem como os métodos de comunicação entre eles.
- **Visão de Implantação**: Descreve a infraestrutura necessária para a execução do sistema, incluindo servidores, rede e dispositivos físicos envolvidos.
- **Visão da Implementação**: Expõe a estrutura do código-fonte, a divisão em camadas e os componentes fundamentais da arquitetura.
- **Visão de Dados (opcional)**: Apresenta o modelo de armazenamento de dados, se aplicável, detalhando como as informações são organizadas e mantidas.
- **Tamanho e Desempenho**: Define as expectativas de escalabilidade e os requisitos de desempenho do sistema.
- **Qualidade**: Descreve os atributos de qualidade do sistema, como confiabilidade, extensibilidade e segurança.

Este documento servirá como referência para a equipe de desenvolvimento e demais stakeholders, garantindo uma visão clara da arquitetura e facilitando futuras manutenções e evoluções do sistema.

2. Representação Arquitetural

O **Sistema de Gerenciamento de Usuários do Cartório da EBAC** é uma aplicação simples baseada em **arquitetura monolítica** e desenvolvida em linguagem C. Ele opera por meio de **interação via terminal**, onde o usuário pode registrar, consultar e excluir dados de usuários armazenados como arquivos individuais no sistema de arquivos.

A arquitetura do sistema pode ser analisada por meio das seguintes visões:

Visão de Casos de Uso

O sistema apresenta três casos de uso principais:

1. **Registrar Usuário** – O usuário insere o CPF, nome, sobrenome e cargo, e esses dados são armazenados em um arquivo cujo nome corresponde ao CPF.
2. **Consultar Usuário** – O usuário fornece um CPF e o sistema busca o arquivo correspondente para exibir os dados armazenados.
3. **Deletar Usuário** – O usuário fornece um CPF e, caso o arquivo correspondente exista, ele é excluído do sistema.

Visão Lógica

O sistema é estruturado de forma procedural, sem a utilização de conceitos de orientação a objetos. Ele é composto por:

- **Funções principais**: `registro()`, `consulta()`, `deletar()`, que implementam as operações essenciais.
- **Função principal (`main()`)**: Responsável por exibir o menu interativo e chamar as funções conforme a escolha do usuário.
- **Manipulação de arquivos**: Os dados são armazenados em arquivos de texto com o nome do CPF do usuário.

Visão de Processos

- O sistema opera de forma **sequencial e síncrona**, onde cada operação é executada individualmente a partir da entrada do usuário.
- Não há concorrência ou execução paralela, pois cada ação é tratada de maneira isolada.

Visão de Implantação

- O sistema é projetado para **executar localmente** em um ambiente Windows, utilizando comandos de terminal (`system("cls")`, `system("pause")`).
- Os dados dos usuários são armazenados no sistema de arquivos do computador como arquivos de texto, em vez de um banco de dados tradicional.
- O código é estruturado em um único arquivo-fonte `.c`, sem modularização.
- A manipulação de arquivos é feita diretamente através das funções `fopen()`, `fprintf()`, `fgets()` e `remove()`.
- A interação com o usuário ocorre via terminal, utilizando `printf()` e `scanf()`.

Essa arquitetura é funcional para aplicações simples, mas apresenta limitações em termos de escalabilidade, persistência de dados e segurança. Caso queira evoluir o sistema, seria interessante considerar o uso de **bancos de dados**, modularização do código e talvez até uma abordagem **orientada a objetos**.

3. Metas e Restrições da Arquitetura

Requisitos e Objetivos do Software

O **Sistema de Gerenciamento de Usuários do Cartório da EBAC** tem como principal objetivo oferecer um meio simples e eficiente para o **registro, consulta e exclusão de usuários** dentro de um ambiente local. Os requisitos e objetivos que impactam sua arquitetura incluem:

Segurança e Privacidade

Os dados dos usuários são armazenados localmente em arquivos de texto nomeados com o CPF do usuário, o que **não oferece criptografia ou controle de acesso**.
Qualquer usuário que tenha acesso ao sistema pode visualizar ou excluir arquivos, representando um risco de **exposição e perda de dados**.
Para aprimorar a segurança, seria necessário implementar **autenticação e controle de permissões**, além de técnicas de criptografia nos arquivos.

Portabilidade e Distribuição

O sistema foi desenvolvido em **linguagem C** e depende de bibliotecas padrão, tornando-o **compatível com diversos sistemas operacionais** (Windows, Linux, macOS), desde que adaptado para remover comandos específicos (`system("cls")` e `system("pause")`).
Por ser um software **monolítico e independente de banco de dados**, a instalação e distribuição são **simples**, exigindo apenas a execução do binário compilado.

Reutilização e Manutenção

O código-fonte **não é modularizado**, o que dificulta a reutilização e manutenção do software.
Para aumentar a **reutilização**, seria ideal **separar a lógica de negócio em arquivos distintos**, criando funções genéricas para manipulação de arquivos.

Uso de Produtos Desenvolvidos Internamente e Ferramentas

O sistema não depende de **ferramentas ou bibliotecas externas**, tornando sua implementação simples.
O desenvolvimento pode ser realizado em qualquer **IDE compatível com C** (Dev-C++, Code::Blocks, Visual Studio Code, etc.).

Restrições Especiais

Estratégia de Design e Implementação

O sistema segue uma abordagem **procedural**, sem a utilização de conceitos de **orientação a objetos**.
A persistência dos dados é baseada em **arquivos de texto**, o que limita a integridade e segurança das informações armazenadas.

Estrutura da Equipe e Cronograma

Como se trata de um **projeto educacional**, espera-se que seja desenvolvido e compreendido por **estudantes com**

conhecimentos básicos em C.
O código foi escrito de forma simples e direta, priorizando **facilidade de entendimento** em vez de escalabilidade ou robustez.

Código-Fonte Legado

Atualmente, **não há código-fonte legado** a ser mantido ou integrado ao projeto.
Caso o sistema precise ser expandido, será necessário reestruturar a arquitetura para suportar novas funcionalidades, como um banco de dados ou uma interface gráfica.

4. Visão de Casos de Uso

Cadastro de Usuário (Registrar Usuário)

Descrição:

O usuário insere seus dados (CPF, nome, sobrenome e cargo), que são então armazenados em um **arquivo de texto** nomeado com o CPF informado.

Fluxo Principal:

1. O sistema solicita o CPF do usuário.
2. O usuário insere o CPF.
3. O sistema cria um **arquivo de texto** com o nome correspondente ao CPF.
4. O sistema solicita o nome, sobrenome e cargo.
5. O usuário insere as informações.
6. O sistema grava os dados no arquivo e finaliza o processo.

Cobertura Arquitetural:

- **Manipulação de Arquivos:** Uso de `fopen()`, `fprintf()`, e `fclose()` para criar e armazenar informações.
- **Interação com o Usuário:** Uso de `printf()` e `scanf()` para coletar e exibir dados.
- **Estratégia de Persistência:** Utilização do **sistema de arquivos** como meio de armazenamento.

Consulta de Usuário

O sistema permite que um usuário consulte informações cadastradas inserindo um CPF válido.

Fluxo Principal:

1. O sistema solicita um CPF.
2. O usuário insere o CPF.
3. O sistema **verifica se o arquivo correspondente existe**.
    a. Se o arquivo **não existir**, exibe uma mensagem de erro.
    b. Se o arquivo **existir**, abre o arquivo e lê os dados armazenados.

4. O sistema exibe as informações do usuário na tela.

Cobertura Arquitetural:

- **Validação de Entrada:** O sistema verifica se o CPF informado corresponde a um arquivo existente.
- **Leitura de Arquivos:** Uso de `fopen()` no modo **leitura (`"r"`)** para buscar informações.
- **Interação com o Usuário:** Exibição dos dados na tela com `printf()`.
- **Tratamento de Erros:** Se o arquivo não existir, o sistema informa que o usuário não está cadastrado.

Exclusão de Usuário

Descrição:

O sistema permite que um usuário exclua um cadastro inserindo um CPF válido.

Fluxo Principal:

1. O sistema solicita um CPF.
2. O usuário insere o CPF.
3. O sistema verifica se o arquivo correspondente existe.
    a. Se **não existir**, exibe uma mensagem informando que o usuário não foi encontrado.
    b. Se **existir**, exclui o arquivo do sistema.
4. O sistema confirma a exclusão para o usuário.

Cobertura Arquitetural:

- **Verificação da Existência do Arquivo:** O sistema tenta abrir o arquivo para verificar se ele existe.
- **Remoção de Arquivos:** Uso da função `remove()` para excluir o arquivo do sistema.
- **Tratamento de Erros:** O sistema exibe mensagens apropriadas caso o CPF não esteja cadastrado.

4.1 Realizações de Casos de Uso

Cadastro de Usuário – Fluxo Completo

Cenário: Um novo usuário deseja se cadastrar no sistema.

Passos Executados:

1. O sistema exibe a opção **"Registrar Usuário"** no menu principal.
2. O usuário seleciona a opção de cadastro (**Entrada via scanf**).
3. O sistema solicita que o usuário insira o **CPF**.

4. O sistema **cria um arquivo nomeado com o CPF informado** e armazena os dados do usuário.
5. O sistema solicita o **Nome, Sobrenome e Cargo**.
6. O usuário insere os dados, que são **salvos no arquivo de texto correspondente ao CPF**.
7. O sistema finaliza o processo e exibe uma confirmação.

Elementos do Modelo de Design Utilizados:

- **Interação com o usuário:** `printf()` para exibir mensagens e `scanf()` para capturar informações.
- **Persistência de Dados:** `fopen()` e `fprintf()` para criação e escrita de arquivos.
- **Separação de Dados:** Cada campo é separado por vírgulas (`,`), permitindo futuras consultas estruturadas.

Consulta de Usuário – Fluxo Completo

Cenário: Um usuário deseja consultar um cadastro existente.

Passos Executados:

1. O sistema exibe a opção **"Consultar Usuário"** no menu principal.
2. O usuário seleciona a opção de consulta e insere um **CPF**.
3. O sistema verifica se o arquivo correspondente ao CPF **existe**.
    a. Se o arquivo **não existir**, exibe uma mensagem de erro.
    b. Se o arquivo **existir**, abre o arquivo no **modo leitura (`"r"`)**.
4. O sistema lê os dados armazenados e exibe as informações do usuário na tela.
5. O sistema finaliza a operação e retorna ao menu.

Elementos do Modelo de Design Utilizados:

- **Validação de Entrada:** O sistema verifica se o arquivo com o CPF informado existe antes de tentar lê-lo.
- **Leitura de Dados:** `fgets()` captura os dados do arquivo e os exibe na tela.
- **Tratamento de Erros:** Se o arquivo não for encontrado, o sistema evita falhas e informa ao usuário.

Exclusão de Usuário – Fluxo Completo

Cenário: O administrador deseja remover um usuário do sistema.

Passos Executados:

1. O sistema exibe a opção **"Deletar Usuário"** no menu principal.
2. O usuário seleciona a opção de exclusão e insere um **CPF**.
3. O sistema verifica se o arquivo correspondente ao CPF **existe**.
    a. Se **não existir**, exibe uma mensagem de erro.
    b. Se **existir**, fecha o arquivo e **remove-o do sistema**.
4. O sistema confirma a remoção do usuário e retorna ao menu principal.

Elementos do Modelo de Design Utilizados:

- **Validação da Existência do Arquivo:** Antes da remoção, o sistema verifica se o CPF cadastrado possui um arquivo correspondente.
- **Exclusão de Dados:** Uso da função `remove()` para deletar o arquivo.
- **Interação com o Usuário:** Exibição de mensagens de confirmação ou erro.

5. Visão Lógica

Visão Lógica – Modelo de Design
*A arquitetura do sistema segue um modelo **modular**, onde cada funcionalidade é separada em **subsistemas independentes**, facilitando a manutenção e a escalabilidade. O sistema é estruturado em um modelo **baseado em arquivos**, utilizando a **linguagem C** para manipulação direta de dados sem banco de dados relacional.*

Divisão em Subsistemas e Pacotes

O software pode ser dividido nos seguintes subsistemas:

1. **Interface com o Usuário** – Gerencia a interação do usuário por meio do terminal.
2. **Gerenciamento de Arquivos** – Responsável pela criação, leitura, escrita e exclusão de arquivos que armazenam os dados dos usuários.
3. **Operações de Dados** – Estrutura e manipula os dados, garantindo que os registros sejam armazenados e recuperados corretamente. Cada subsistema é implementado através de funções específicas no código.

Estrutura de Classes e Utilitários

Em C, não há um conceito formal de classes como em linguagens orientadas a objetos (Java, Python, C++), mas podemos representar a estrutura do sistema em **módulos (funções separadas)**, que atuam de maneira semelhante a **classes** em sistemas orientados a objetos.

Pacote 1: Interface com o Usuário

Este pacote gerencia a entrada e saída de dados com o usuário. Ele contém:

- **Função `main()`**
  - Responsável por exibir o menu principal e capturar a escolha do usuário.
  - Direciona para os métodos correspondentes com base na opção selecionada.
  - Controla o loop de execução do sistema.

Pacote 2: Gerenciamento de Arquivos

Este pacote trata da manipulação dos arquivos que armazenam os registros de usuários. Ele contém:

- **Função `registro()`**
  - Responsável pela criação e escrita de um arquivo para cada usuário.
  - Cada arquivo é nomeado com o CPF do usuário.
  - Os dados são armazenados no formato CSV (valores separados por vírgula).
- **Função `consulta()`**
  - Lê o arquivo correspondente ao CPF informado pelo usuário.
  - Exibe os dados armazenados no arquivo na tela.
  - Trata erros caso o arquivo não seja encontrado.
- **Função `deletar()`**
  - Verifica se o arquivo do usuário existe.
  - Caso exista, remove o arquivo do sistema.
  - Retorna mensagens de erro ou sucesso ao usuário.

Pacote 3: Operações de Dados

Este pacote define como os dados são processados e organizados. Ele contém:

Manipulação de Strings (`<string.h>`)

- Usa `fprintf()` para salvar os dados de forma organizada dentro dos arquivos.
- Utiliza `strcpy()` para copiar informações para variáveis.

Armazenamento Estruturado

- Cada dado (CPF, nome, sobrenome, cargo) é separado por vírgula para facilitar futuras operações de leitura.
- `fgets()` é usado para recuperar os dados armazenados e exibi-los no terminal.

Relacionamentos e Dependências

- O **subsistema de Interface** se comunica diretamente com o **subsistema de Gerenciamento de Arquivos**, chamando suas funções (`registro()`, `consulta()`, `deletar()`).
- O **subsistema de Gerenciamento de Arquivos** depende do **subsistema de Operações de Dados** para manipular corretamente os registros.
- A comunicação entre subsistemas ocorre por meio da **troca de informações via argumentos de função**, sem uso de banco de dados ou memória dinâmica.

Principais Operações e Atributos

Main(): Controla a interface com o usuário e direciona para as operações adequadas.
Exibe menu, captura entrada, chama funções de cadastro, consulta e exclusão.

Registro(): Cria um arquivo para armazenar os dados de um usuário. Suas operações importantes são; fopen(), fprintf(), fclose(), strcpy().

Consulta(): Lê os dados do arquivo correspondente ao CPF informado. Suas operações importantes no sistema são; fopen(), fgets(), fclose().

Deletar() : Remove o arquivo do usuário se ele existir. E suas respectivas operações relevantes no sistema são; fopen() e remove ().

5.1 Visão Geral

Estrutura em Camadas

O software pode ser decomposto nas seguintes camadas:

Camada de Apresentação (Interface com o Usuário)

Responsável pela interação com o usuário, capturando entrada de dados e exibindo saídas.
Implementada pela **função `main()`**, que exibe o menu e direciona o fluxo do programa.

Camada de Aplicação

Gerencia as operações principais do sistema, como registro, consulta e exclusão de usuários.
Implementada pelas funções:
`registro()` – Criação e armazenamento de usuários.
`consulta()` – Recuperação e exibição de usuários cadastrados.
`deletar()` – Exclusão de registros de usuários.

Camada de Persistência de Dados

Responsável pela manipulação dos arquivos que armazenam os dados dos usuários.
Utiliza as funções de manipulação de arquivos da biblioteca padrão do C:
`fopen()`, `fclose()`, `fprintf()`, `fgets()`, `remove()`.

Hierarquia de Pacotes

Embora a linguagem C não tenha pacotes no sentido tradicional de linguagens orientadas a objetos, podemos estruturar o código em **módulos** lógicos, organizados conforme suas responsabilidades:

Pacote 1: Interface do Usuário

Arquivo: `main()`
Responsabilidade:
Apresenta o menu ao usuário.
Captura a opção escolhida.
Direciona para a função correspondente (`registro()`, `consulta()`, `deletar()`).

Pacote 2: Gerenciamento de Usuários

Funções: `registro()`, `consulta()`, `deletar()`.
Responsabilidade:
Registrar novos usuários no sistema.
Consultar os dados de um usuário pelo CPF.
Excluir um usuário e seus dados do sistema.

Pacote 3: Manipulação de Arquivos

Funções Utilizadas:
`fopen()` – Abre arquivos para leitura/escrita.
`fprintf()` – Escreve dados em arquivos.
`fgets()` – Lê dados armazenados.
`remove()` – Exclui arquivos.

Responsabilidade:
Criar e armazenar arquivos de usuários.
Manipular o conteúdo dos arquivos.
Excluir arquivos quando necessário.

Fluxo de Comunicação entre Camadas

O usuário interage com a Camada de Apresentação (menu principal).
A Camada de Aplicação recebe a solicitação do usuário e direciona para a funcionalidade correspondente.
A Camada de Persistência manipula os arquivos para armazenar, recuperar ou excluir os dados.
O resultado é retornado à Camada de Apresentação, que exibe a saída ao usuário.

Esse modelo permite que futuras mudanças, como a migração para um banco de dados, sejam feitas sem afetar toda a estrutura do software.

5.2 Pacotes de Design Significativos do Ponto de Vista da Arquitetura

Pacote 1: Interface do Usuário

Este pacote gerencia a interação com o usuário, apresentando o menu e capturando as opções escolhidas.

Classes Significativas:

Classe Main
Descrição: Responsável pelo controle do fluxo principal do sistema.
Principais operações:
exibirMenu(): Exibe as opções disponíveis para o usuário.
capturarOpcao(): Recebe a entrada do usuário e direciona para a funcionalidade correspondente.

Pacote 2: Gerenciamento de Usuários

Este pacote contém as funcionalidades principais do sistema, incluindo registro, consulta e exclusão de usuários.

Classes Significativas:

Classe RegistroUsuario

Descrição: Gerencia o cadastro de usuários.
Principais operações:
registrar(cpf, nome, sobrenome, cargo): Cria um novo registro de usuário e o armazena em um arquivo.

Classe ConsultaUsuario

Descrição: Permite a consulta de usuários cadastrados no sistema.
Principais operações:
consultar(cpf): Busca o arquivo correspondente ao CPF e exibe as informações do usuário.

Classe DeletarUsuario

Descrição: Gerencia a exclusão de usuários cadastrados.
Principais operações:
deletar(cpf): Remove o arquivo correspondente ao CPF do sistema.

Pacote 3: Manipulação de Arquivos

Este pacote gerencia a persistência de dados do sistema, lidando diretamente com a criação, leitura e exclusão de

arquivos.

Classes Significativas:

Classe GerenciadorArquivos
Descrição: Responsável pelo acesso e manipulação dos arquivos no sistema.
Principais operações:
abrirArquivo(cpf, modo): Abre um arquivo no modo especificado (leitura, escrita, atualização).
escreverNoArquivo(dados): Escreve informações no arquivo correspondente.
lerArquivo(cpf): Retorna os dados armazenados no arquivo do usuário.
removerArquivo(cpf): Exclui o arquivo associado ao CPF informado.

6. Visão de Processos

Processos do Sistema

O sistema pode ser dividido logicamente em **três grupos de processos principais**, que interagem de maneira sequencial:

Processo de Interface do Usuário (Processo Principal)

- **Descrição:** Esse processo gerencia a interação do usuário, exibindo o menu e capturando a escolha para redirecionar a execução ao processo correspondente.
- **Tipo:** Processo **pesado** (único fluxo de controle que executa subprocessos conforme a escolha do usuário).
- **Atividades:**
    - Exibição do menu principal.
    - Recebimento da entrada do usuário.
    - Chamada dos processos de registro, consulta ou exclusão de usuários.

Processo de Manipulação de Dados (Subprocesso Leve)

- **Descrição:** Este processo é acionado sempre que uma operação envolvendo arquivos é necessária, como criar, ler ou excluir registros. Ele lida com a persistência dos dados do usuário.
- **Tipo:** Processo **leve** (subprocesso do processo principal).
- **Atividades:**
  Criar arquivos para novos registros.
  Ler arquivos ao consultar usuários.
  Excluir arquivos ao deletar um usuário.
- **Interação:** O processo recebe comandos do **Processo de Interface do Usuário**, executa a ação correspondente e retorna um status (sucesso ou erro).

Processo de Controle de Fluxo (Subprocesso Leve)

- **Descrição:** Responsável por garantir que as operações do sistema ocorram de forma ordenada, evitando

execução incorreta ou interrupções inesperadas.
- **Tipo:** Processo **leve** (subprocesso auxiliar do fluxo principal).
- **Atividades:**
  Verificar se um CPF já está cadastrado antes de criar um novo usuário.
  Exibir mensagens de erro quando um usuário não é encontrado.
  Garantir que a execução ocorra sem interrupções inesperadas.
- **Interação:** Este processo se comunica com o **Processo de Manipulação de Dados**, validando a existência de registros e garantindo a integridade dos dados.

Comunicação entre Processos

A comunicação entre os processos ocorre de forma **sincronizada e sequencial**, utilizando os seguintes mecanismos:

Comunicação por Chamadas Diretas (Sincronização Sequencial)

- O **Processo de Interface do Usuário** chama diretamente os subprocessos de **Manipulação de Dados** e **Controle de Fluxo** quando necessário.
- O fluxo segue um padrão linear:
  - Usuário seleciona uma opção no menu.
  - O sistema processa a escolha.
  - O resultado é retornado e exibido.

Armazenamento Temporário (Arquivos como Meio de Comunicação)

- O sistema utiliza **arquivos** como meio de comunicação entre os processos. O **Processo de Manipulação de Dados** escreve e lê arquivos para armazenar informações, garantindo que os dados persistam entre diferentes execuções do sistema.

Mensagens de Erro e Interrupções

- O **Processo de Controle de Fluxo** gera mensagens de erro quando há falhas (exemplo: tentativa de deletar um usuário inexistente).
- O sistema não utiliza **interrupções assíncronas** ou manipulação avançada de threads, mantendo uma execução **determinística e controlada**.

7. Visão de Implementação

A implantação do software ocorre em um ambiente de **rede local (LAN)** ou em computadores individuais, dependendo da necessidade do escritório. A seguir, detalhamos as configurações da rede física e a distribuição dos processos dentro do sistema.

Configuração de Rede Física e Harware

O software pode ser executado em diferentes cenários, de acordo com a infraestrutura do escritório fictício da

EBAC:

Cenário 1: Execução Local (Standalone)

- **Descrição:** O sistema é instalado e executado diretamente em computadores individuais sem necessidade de comunicação em rede.
- **Nós físicos:** Computadores pessoais (PCs ou notebooks).
- **Configuração recomendada:**
  Processador: Intel Core i3 (ou superior) / AMD Ryzen 3 (ou superior).
  Memória RAM: Mínimo de 4GB.
  Armazenamento: HDD ou SSD com pelo menos 500MB disponíveis.
  Sistema Operacional: Windows, Linux ou macOS.
  Interconexão: Nenhuma, pois os dados são armazenados localmente.
- **Uso:** Pequenos escritórios onde cada máquina opera de forma independente.

Cenário 2: Rede Local (LAN) – Compartilhamento de Arquivos

- **Descrição:** O software é implantado em uma rede interna, onde vários computadores acessam um servidor central que armazena os arquivos dos usuários.
- **Nós físicos:**
  Servidor de Arquivos: Máquina central para armazenamento dos registros.
  Estações de Trabalho: Computadores conectados à rede interna.
- **Configuração recomendada:**
  Servidor:
  Processador Intel Xeon / AMD Ryzen 7 ou superior.
  16GB de RAM ou mais.
  SSD de pelo menos 1TB.
  Sistema Operacional Linux Server ou Windows Server.
  Estações de Trabalho: Configuração mínima igual ao cenário standalone.
  Interconexão:
  Comunicação via **LAN** (cabo Ethernet Cat5e/Cat6 ou Wi-Fi de 5GHz).
  Acesso via **Protocolo SMB/NFS** para compartilhamento dos arquivos.
  Uso: Escritórios que desejam centralizar os registros e permitir múltiplos acessos.

Mapeamento dos Processos na Infraestrutura Física

- O software executa três principais processos, distribuídos conforme a configuração da rede:

  Registro de Usuário: Criado e salvo no disco local. Criado e armazenado no servidor de arquivos.
  Consulta de Usuário: Lido do disco local. Acessado remotamente via LAN.
  Exclusão de Usuário: Removido do disco local. Arquivo deletado no servidor de arquivos

  No **modo standalone**, todas as operações ocorrem no próprio computador do usuário.
  No **modo LAN**, as estações de trabalho enviam solicitações ao servidor central, que executa as operações de leitura, escrita e exclusão.

8. Visão da Implementação

Estrutura Geral do Modelo de Implementação

O sistema segue uma abordagem estruturada e se divide nos seguintes componentes principais:

Camada de Interface do Usuário

Descrição: Gerencia a interação com o usuário por meio de um menu textual.
Principais Elementos:
`main()`: Exibe o menu e recebe a entrada do usuário.
`printf() / scanf()`: Funções padrão para exibição e leitura de entrada do usuário.
`system("cls")`: Limpa a tela para melhor visualização.

Camada de Lógica de Negócio

Descrição: Contém as regras de negócio e o processamento dos dados do usuário.
Principais Elementos:
`registro()`: Captura os dados do usuário e os salva em arquivos.
`consulta()`: Recupera e exibe os dados do usuário com base no CPF informado.
`deletar()`: Remove os registros de usuários do sistema.

Camada de Persistência de Dados

Descrição: Responsável pela leitura, escrita e exclusão de arquivos que armazenam os registros dos usuários.
Principais Elementos:
`FILE *file`: Representa um arquivo no sistema.
`fopen() / fclose()`: Abre e fecha arquivos para leitura/escrita.
`fprintf() / fgets() / remove()`: Manipulam o conteúdo dos arquivos.

Modelo de Implementação e Organização de Código

O sistema é estruturado em um único arquivo fonte (`main.c`), com funções separadas para cada operação. A estrutura modular permite que cada funcionalidade seja chamada conforme a necessidade, tornando o código mais organizado e fácil de entender.

8.1 Visão Geral

Componentes Significativos da Arquitetura

Os principais componentes do sistema incluem:

Estruturas de Dados e Arquivos

Os dados do usuário são armazenados em arquivos de texto individuais, nomeados pelo CPF do

usuário.
Cada arquivo contém as informações separadas por vírgulas, Ex:

```
12345678900,João,Silva,Gerente
```

Fluxo de Dados e Processamento

O **main()** gerencia a navegação do usuário e chama a função apropriada com base na escolha.
As funções **registro()**, **consulta()** e **deletar()** manipulam os arquivos diretamente para armazenar ou recuperar informações.
O sistema utiliza verificações básicas para garantir que os dados sejam recuperados corretamente.

8.2 Camadas

Camada de Interface do Usuário

Responsabilidade: Gerenciar a interação entre o usuário e o sistema, capturando entradas e exibindo saídas.

Subsistemas:

- **Menu Principal:** Apresenta as opções disponíveis e recebe a escolha do usuário.
- **Saída de Dados:** Exibe mensagens e informações processadas ao usuário.
- **Entrada de Dados:** Captura os inputs do usuário para envio à camada de lógica de negócio.

Diagrama de Componentes:

```
+----------------------------------+
| Interface do Usuário             |
|----------------------------------|
| - Exibe menu principal           |
| - Captura entrada do usuário     |
| - Exibe resultados               |
+----------------------------------+
```

Camada de Lógica de Negócio

Responsabilidade: Processar as informações fornecidas pelo usuário e aplicar as regras do sistema.

Subsistemas:

- **Registro de Usuário:** Processa e formata os dados para armazenamento.
- **Consulta de Usuário:** Busca e organiza as informações para exibição.
- **Exclusão de Usuário:** Verifica e deleta os registros armazenados.

Diagrama de Componentes:

```
+---------------------------------+
| Lógica de Negócio               |
|---------------------------------|
| - Processa registro de usuários  |
| - Processa consulta de usuários  |
| - Processa exclusão de usuários  |
+---------------------------------+
```

Camada de Persistência de Dados

Responsabilidade: Armazenar e recuperar os dados dos usuários em arquivos no sistema.

Subsistemas:

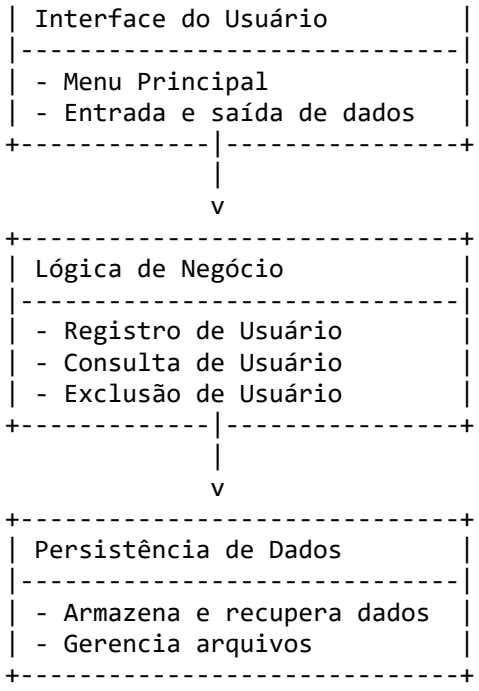
- **Gerenciamento de Arquivos:** Manipulação e organização dos arquivos no sistema.
- **Escrita de Dados:** Grava as informações do usuário nos arquivos.
- **Leitura de Dados:** Recupera informações de arquivos existentes.
- **Remoção de Dados:** Exclui registros permanentemente.

Diagrama de Componentes:

```
+---------------------------------+
| Persistência de Dados           |
|---------------------------------|
| - Grava dados em arquivos       |
| - Lê dados armazenados          |
| - Remove arquivos               |
+---------------------------------+
```

Diagrama Geral do Sistema

```
+----------------------------+
```



9. Tamanho e Desempenho

O sistema foi projetado para ser leve e eficiente, garantindo a execução fluida mesmo em **ambientes com recursos limitados**. Abaixo, estão as principais características de **dimensionamento** e **restrições de desempenho** que influenciam a arquitetura do software.

Características de Dimensionamento

Armazenamento baseado em arquivos:

- Cada usuário é registrado em um arquivo de texto separado, utilizando o CPF como identificador único.
- O crescimento do número de usuários impacta diretamente a quantidade de arquivos no diretório.
- Para cenários com um grande volume de usuários, pode ser necessário migrar para um banco de dados relacional.

Tamanho dos Arquivos:

- Como cada arquivo contém apenas um registro de usuário, o tamanho individual dos arquivos é pequeno.
- O consumo de espaço em disco é mínimo, mas pode crescer com o aumento do número de registros.

Escalabilidade:

- O modelo de armazenamento atual é eficiente para um número pequeno ou médio de usuários.

- Em um ambiente com milhares de usuários, a busca e a manipulação de arquivos podem se tornar mais lentas.
- Alternativas como indexação ou banco de dados seriam necessárias para otimização.

Restrições de Desempenho

Tempo de Resposta:

- O sistema responde rapidamente para um número limitado de usuários devido à leitura direta de arquivos.
- Em um volume muito grande, a performance da busca e remoção pode ser afetada devido à falta de indexação eficiente.

Processamento Simultâneo:

- O código atual não suporta **execução concorrente**.
- Se múltiplos usuários tentarem acessar o sistema ao mesmo tempo, pode ocorrer corrupção ou bloqueio de arquivos.
- Implementação de um sistema de bloqueio ou um banco de dados poderia melhorar essa limitação.

Uso de Memória:

- Como o programa só carrega um arquivo na memória por vez, o consumo de RAM é mínimo.
- A eficiência da memória depende da implementação de buffers na leitura e escrita de arquivos.

Portabilidade:

- O sistema pode ser executado em qualquer máquina com um compilador C e suporte a manipulação de arquivos.
- Por ser baseado em arquivos locais, ele **não depende da internet** ou de servidores externos.

10. Qualidade

A arquitetura do software foi projetada para garantir não apenas a funcionalidade do sistema, mas também aspectos essenciais como **extensibilidade, confiabilidade, portabilidade, segurança e privacidade**.

Extensibilidade

A estrutura modular do software permite fácil adaptação para futuras melhorias.

- O sistema atual é baseado em **funções independentes** para registro, consulta e exclusão de usuários, tornando mais simples a adição de novas funcionalidades sem impactar o código existente.
- Caso seja necessário migrar para um **banco de dados** no futuro, a estrutura de leitura/escrita em arquivos pode ser substituída por chamadas SQL sem modificar toda a lógica do sistema.
- Novos campos de usuário podem ser adicionados sem grandes mudanças na estrutura geral do código.

Confiabilidade

O sistema foi desenvolvido para minimizar falhas e garantir o funcionamento correto.

- **Verificação de existência de arquivos:** Antes de acessar ou excluir um usuário, o programa verifica se o arquivo correspondente existe, evitando erros.
- **Tratamento de erros:** Caso um CPF não seja encontrado, o sistema exibe uma mensagem informando que o usuário não foi localizado.
- **Uso de arquivos independentes:** Cada usuário é armazenado separadamente, evitando perda massiva de dados em caso de falha.

Portabilidade

O software foi desenvolvido em **linguagem C**, garantindo compatibilidade com diferentes plataformas.

- Pode ser compilado e executado em **Windows, Linux e macOS** sem grandes alterações no código.
- Não depende de bibliotecas externas complexas, tornando sua instalação e execução simples.
- Como utiliza apenas arquivos locais, pode funcionar **offline**, sem necessidade de conexão com servidores externos.

Segurança e Privacidade

A arquitetura do software busca garantir a proteção dos dados armazenados.

- **Acesso restrito por CPF:** Cada usuário é identificado exclusivamente por seu CPF, evitando duplicações e garantindo registros únicos.
- **Armazenamento local:** Os dados dos usuários são armazenados diretamente no sistema, sem exposição a redes externas, reduzindo o risco de ataques cibernéticos.
- **Privacidade dos dados:** Apenas usuários com acesso ao sistema podem consultar os registros, garantindo que informações pessoais não fiquem expostas.
- **Possibilidade de implementação de criptografia:** Em futuras versões, pode-se adicionar criptografia nos arquivos para maior proteção dos dados armazenados.