UNIVERSIDADE PAULISTA – UNIP

PROJETO INTEGRADO MULTIDICIPLINAR

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Isaque de Medeiros Pinto -R8189G2

Rafael Ferreira Da Cruz –H7565H3

Yuri Barbosa Borges – R874HI9

PROJETO INTEGRADO MULTIDISCIPLINAR II

SÃO PAULO

2025

UNIVERSIDADE PAULISTA

Isaque de Medeiros Pinto – R8189G2

Rafael Ferreira Da Cruz –H7565H3

Yuri Barbosa Borges – R874HI9

PROJETO INTEGRADO MULTIDISCIPLINAR II

Projeto Integrado Multidisciplinar apresentado à Universidade Paulista para a obtenção do título de tecnólogo(a) em Análise e Desenvolvimento de Sistemas.

Orientadora: Prof.ª Larissa Damiani.

SÃO PAULO

2025

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema acadêmico inteligente com apoio de Inteligência Artificial para instituições de ensino. O sistema foi projetado para gerenciar turmas, alunos, aulas e atividades de forma colaborativa, substituindo processos manuais baseados em papel por soluções digitais integradas. Utilizando linguagens Python e C, o sistema implementa funcionalidades de cadastro, matrícula, registro de frequência, lançamento de notas e geração de relatórios. A aplicação de engenharia de software ágil garantiu o ciclo de vida do projeto, enquanto recursos de IA auxiliaram no desenvolvimento de algoritmos e otimização de código. O sistema promove a sustentabilidade ambiental através da significativa redução do uso de papel e oferece uma solução eficiente para a gestão acadêmica institucional.

Palavras-chave: Sistema Acadêmico, Inteligência Artificial, Python, Sustentabilidade, Gestão Educacional.

ABSTRACT

This work presents the development of an intelligent academic system with Artificial Intelligence support for educational institutions. The system was designed to manage classes, students, classes and activities in a collaborative way, replacing manual paper-based processes with integrated digital solutions. Using Python and C languages, the system implements registration, enrollment, attendance recording, grade posting and report generation functionalities. The application of agile software engineering ensured the project life cycle, while AI resources assisted in algorithm development and code optimization. The system promotes environmental sustainability through significant paper use reduction and offers an efficient solution for institutional academic management.

Keywords: System, Artificial Intelligence, Python, Sustainability, Educational Management.

SUMÁRIO

1.1 Contextualização

Uma instituição de ensino necessita de um sistema colaborativo para apoiar professores e alunos no gerenciamento de turmas, aulas e atividades. Atualmente, controles são realizados de forma descentralizada (planilhas, e-mails, mensagens em aplicativos). O sistema deve permitir cadastro de turmas e de alunos, registro de aulas, diário eletrônico de atividades, upload e consulta de atividades, fila de atendimento do aluno na secretaria, entre outras funcionalidades.

1.2 Objetivo Geral

O objetivo geral deste projeto é projetar e implementar um sistema acadêmico colaborativo que permita o cadastro de turmas e alunos, registro de aulas, diário eletrônico de atividades, upload e consulta de atividades, e fila de atendimento do aluno na secretaria, explorando práticas de engenharia de software e recursos de Inteligência Artificial.

1.3 Objetivos específicos

- Aplicar engenharia de software ágil para elaborar os requisitos e apresentar o ciclo de vida do projeto.
- Criar modelos de análise e projeto de sistemas, incluindo diagramas UML de caso de uso.
- Implementar, no sistema acadêmico, algoritmos e estruturas de dados em Python e/ou em C.
- Usar a Inteligência Artificial no desenvolvimento, apresentando, no trabalho acadêmico, os prompts utilizados.
- Elaborar um manual do usuário, explicando as funcionalidades do sistema, com prints de tela do sistema rodando acompanhados de texto explicativo.
- Explicar como a utilização do sistema pode reduzir a utilização de papel na instituição de ensino, trazendo dicas de educação ambiental.
- Elaborar o diagrama da rede local (LAN) da instituição de ensino que utilizará o sistema acadêmico.

- **2 SOBRE A INSTITUIÇÃO DE ENSINO Paulista (IETEP)**

O Instituto de Educação Tecnológica Paulista (IETEP) é uma instituição fictícia especializada em educação profissional e tecnológica, localizada na cidade de São Paulo. A instituição atende aproximadamente 600 alunos em cursos técnicos e tecnológicos.

2.1 Cursos e Estrutura

O IETEP oferece os seguintes cursos: Análise e Desenvolvimento de Sistemas, Redes de Computadores, Gestão da Tecnologia da Informação e Desenvolvimento de Software. A infraestrutura física compreende: 3 laboratórios de informática, 8 salas de aula teóricas, 1 biblioteca, 1 secretaria acadêmica e 1 sala de professores.

2.2 Relacionamento com a Equipe

A equipe de desenvolvimento foi contratada como consultores externos pela direção do IETEP para desenvolver um sistema acadêmico integrado que atendesse às necessidades de digitalização dos processos institucionais.

Figura 1 – Layout físico do IETEP

Fonte: autoria própria.

- **3 PROJETO DA REDE DOS COMPUTADORES DA INSTITUIÇÃO DE ENSINO**

3.1 O que é o projeto de rede

O projeto de rede de computadores consiste no planejamento e documentação da infraestrutura de comunicação de dados que interconecta todos os equipamentos computacionais da instituição de ensino. Este projeto define como os computadores, servidores, impressoras e outros dispositivos se comunicam entre si e com a internet, garantindo que o sistema acadêmico desenvolvido possa funcionar adequadamente em toda a instituição.

3.2 Componentes da rede

A rede local (LAN) do IETEP foi projetada com os seguintes componentes principais:

· Roteador: Dispositivo que conecta a rede interna à internet

· Switches: Equipamentos que interconectam os computadores dentro da instituição

· Servidor: Computador central que armazena o sistema acadêmico e os dados

· Cabeamento estruturado: Infraestrutura física de conexão (cabos CAT6)

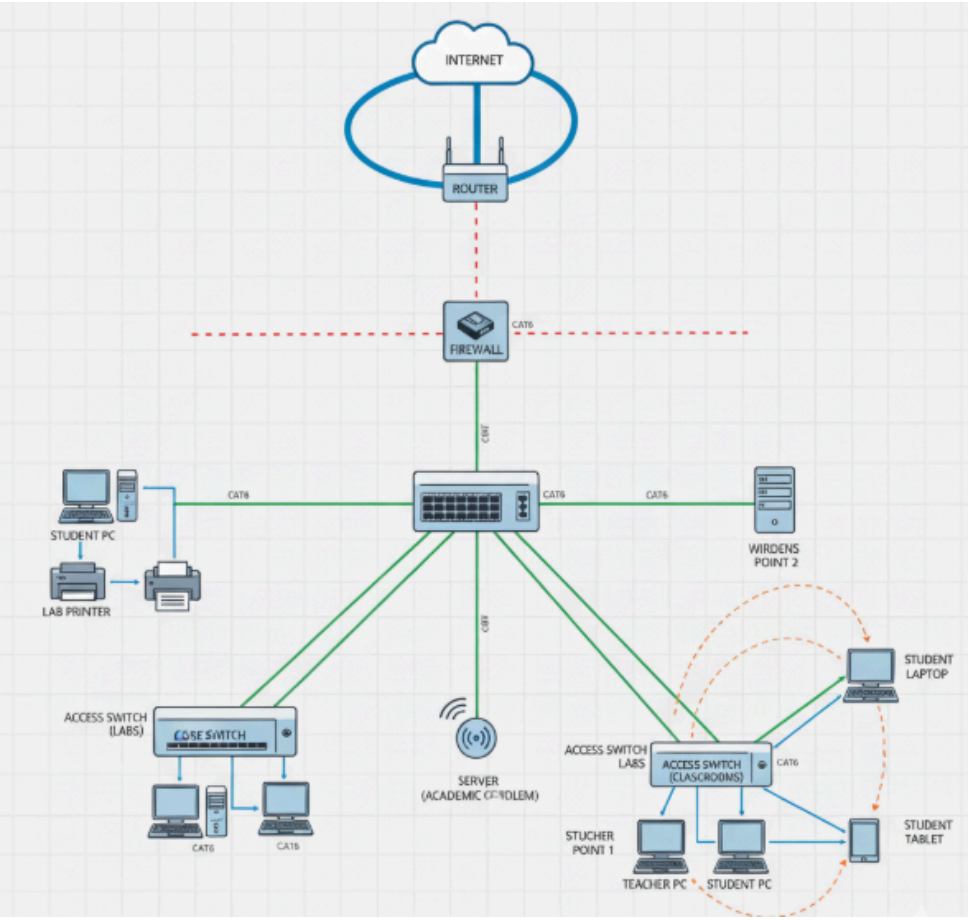
· Access Points: Dispositivos para conexão wireless

3.3 Topologia utilizada

Foi adotada a topologia em estrela, onde todos os dispositivos se conectam a um ponto central (switch). Esta topologia foi escolhida por oferecer melhor desempenho,

facilidade de manutenção e escalabilidade, permitindo que novos equipamentos sejam adicionados sem afetar o funcionamento da rede existente.

Figura 2 – Diagrama da rede local do IETEP



Fonte: autoria própria

- **4 REQUISITOS DE SOFTWARE**

Requisitos de software são descrições detalhadas do que um sistema deve fazer e como deve se comportar. Eles servem como contrato entre os desenvolvedores e os usuários, garantindo que o sistema desenvolvido atenda às necessidades reais da instituição. Os requisitos são divididos em funcionais (o que o sistema faz) e não funcionais (como o sistema faz).

4.1 Modelo de ciclo de vida

O que é modelo de ciclo de vida: É a abordagem metodológica que define as fases de desenvolvimento do software, desde a concepção até a manutenção. Para este projeto, foi adotado o modelo ágil Scrum, que permite desenvolvimento iterativo e incremental, com entregas frequentes de funcionalidades e adaptação contínua aos requisitos.

Justificativa da escolha: O Scrum foi selecionado por permitir:

· Adaptação rápida a mudanças de requisitos

· Entrega contínua de valor para a instituição

· Maior envolvimento dos usuários durante o desenvolvimento

· Detecção precoce de problemas

4.2 Requisitos funcionais

Requisitos funcionais definem as funcionalidades específicas que o sistema deve executar. Eles descrevem "o que" o sistema faz em termos de comportamentos, funções e serviços que devem ser disponibilizados aos usuários.

Sistema Acadêmico IETEP - Requisitos Funcionais:

RF01	Cadastrar turmas	O sistema deve permitir o cadastro de novas turmas com código, disciplina, professor e horário
RF02	Matricular alunos	O sistema deve possibilitar a matrícula de alunos em turmas específicas
RF03	Registrar frequência	O sistema deve permitir o registro de presença e ausência dos alunos
RF04	Lançar notas	O sistema deve aceitar o lançamento de notas por atividade avaliativa
RF05	Gerenciar fila de atendimento	O sistema deve controlar a ordem de atendimento na secretaria
RF06	Upload de atividades	O sistema deve permitir o upload e download de materiais didáticos

RF07	Emitir relatórios	O sistema deve gerar relatórios de frequência e rendimento acadêmico

4.3 Requisitos não funcionais

Requisitos não funcionais definem os critérios de qualidade e restrições do sistema. Eles especificam "como" o sistema deve desempenhar suas funções, abordando aspectos como desempenho, segurança, usabilidade e confiabilidade.

RNF01	Desempenho	Tempo de resposta inferior a 5 segundos para todas as operações
RNF02	Usabilidade	Interface em português com menus intuitivos
RNF03	Confiabilidade	Sistema deve estar disponível 95% do tempo durante o horário comercial
RNF04	Compatibilidade	Funcionamento em Windows 10 e Linux Ubuntu

RNF05	Segurança	Acesso por usuário e senha para módulos sensíveis
RNF06	Manutenibilidade	Código documentado e modular para facilitar futuras alterações

- **5 MODELAGEM UML**

5.1 O que é modelagem UML

A modelagem UML (Unified Modeling Language) é uma linguagem padrão para visualizar, especificar, construir e documentar artefatos de sistemas de software. Ela fornece notações e diagramas padronizados para representar diferentes aspectos do sistema, facilitando a comunicação entre desenvolvedores, usuários e stakeholders.

5.2 Diagrama de Casos de Uso

O que é um diagrama de casos de uso: É um diagrama UML que representa as funcionalidades do sistema sob a perspectiva do usuário. Ele mostra os atores (usuários do sistema) e os casos de uso (funcionalidades) que esses atores podem executar, além dos relacionamentos entre eles.

Figura 3 - Diagrama de casos de uso mostrando as interações entre atores e sistema

Fonte: autoria própria.

5.3 Atores identificados

- Professor: Responsável por registrar frequência, lançar notas e consultar turmas
- Aluno: Pode consultar notas, verificar frequência e acessar materiais
- Secretária: Responsável por cadastrar turmas, matricular alunos e gerar relatórios

5.4 Casos de uso principais

Cada caso de uso representa uma funcionalidade específica do sistema:

- Cadastrar turma
- Matricular aluno
- Registrar frequência
- Lançar notas
- Consultar histórico
- Gerar relatórios
- Gerenciar fila de atendimento

6 SOFTWARE DA INSTITUIÇÃO DE ENSINO

O software desenvolvido é um sistema acadêmico integrado que automatiza os processos educacionais da instituição. Ele substitui sistemas manuais e baseados em papel por soluções digitais, oferecendo módulos específicos para professores, alunos e funcionários da secretaria.

6.1 Manual do software

O que é o manual do software: É um guia de utilização que explica como operar o sistema desenvolvido. Ele descreve as funcionalidades disponíveis em cada módulo, fornece exemplos de uso e ajuda os usuários a tirar o máximo proveito do sistema.

Funcionalidades por módulo:

Módulo Professor:

· Registrar frequência: Permite marcar presença/ausência dos alunos

· Lançar notas: Interface para inserir notas de atividades avaliativas

· Consultar turmas: Visualização das turmas sob responsabilidade do professor

Módulo Secretaria:

· Cadastrar turmas: Criação de novas turmas no sistema

· Matricular alunos: Vinculação de alunos às turmas

· Gerar relatórios: Emissão de documentos institucionais

Módulo Aluno:

· Consultar notas: Acesso às notas e médias

· Ver frequência: Controle de presenças e ausências

· Materiais: Download de conteúdos didáticos

6.2 Uso de inteligência artificial no desenvolvimento

O que é o uso de IA no desenvolvimento: Refere-se à aplicação de ferramentas de inteligência artificial para auxiliar nas diversas fases do desenvolvimento de software, desde a geração de código até a otimização e correção de problemas.

Aplicações específicas da IA:

Otimização de algoritmos:

· Uso de IA para sugerir estruturas de dados mais eficientes

· Análise automática de complexidade de algoritmos

· Sugestões para redução de tempo de execução

Correção de bugs:

· Detecção automática de possíveis erros de lógica

· Sugestões para tratamento de exceções

· Análise de possíveis vazamentos de memória

Geração de código:

· Assistência na criação de funções complexas

· Sugestões de boas práticas de programação

· Ajuda na documentação do código

6.3 Redução do uso de papel

O que é a redução do uso de papel: É a substituição de processos físicos baseados em papel por soluções digitais, eliminando a necessidade de impressão, armazenamento físico e transporte de documentos.

Processos impactados pela digitalização:

· Diários de classe: Eliminação dos diários físicos de frequência e notas

· Fichas de matrícula: Substituição de formulários impressos por cadastro digital

· Relatórios acadêmicos: Geração eletrônica de boletins e históricos

· Comunicações internas: Redução de memorandos e circulares impressas

Benefícios ambientais:

· Redução do consumo de água e energia na produção de papel

· Diminuição da emissão de CO_2 no transporte e produção

· Menor geração de resíduos sólidos

· Preservação de recursos florestais

• 7 CONSIDERAÇÕES FINAIS

O desenvolvimento do Sistema Acadêmico Inteligente para o Instituto de Educação Tecnológica Paulista (IETEP) permitiu a consolidação dos conhecimentos adquiridos nas disciplinas do curso de Análise e Desenvolvimento de Sistemas, demonstrando na prática a aplicação integrada dos conceitos teóricos.

Em relação à Engenharia de Software Ágil, a adoção da metodologia Scrum mostrou-se adequada para o desenvolvimento iterativo e incremental do sistema. A divisão do projeto em sprints permitiu entregas constantes de valor e adaptações

rápidas aos feedbacks recebidos, comprovando a eficácia das metodologias ágeis em projetos de médio porte.

Quanto à Análise e Projeto de Sistemas, a modelagem UML, especialmente o diagrama de casos de uso, foi fundamental para compreender e documentar as interações entre os usuários e o sistema. A modelagem mostrou-se essencial para garantir que todos os requisitos funcionais fossem adequadamente contemplados no projeto.

Na implementação em Python e C, observou-se a complementaridade das duas linguagens. O Python mostrou-se ideal para o desenvolvimento rápido dos módulos principais e manipulação de dados, enquanto o C demonstrou superioridade em operações que requerem maior controle de memória e desempenho, como o módulo de fila de atendimento.

Sobre o uso de Inteligência Artificial, constatou-se que as ferramentas de IA generativa foram valiosas auxiliares no desenvolvimento, especialmente na otimização de algoritmos, sugestão de estruturas de dados e correção de bugs. No entanto, verificou-se que o conhecimento técnico do desenvolvedor permanece essencial para validar e integrar adequadamente as sugestões geradas pela IA.

Em relação à Educação Ambiental, o sistema desenvolvido demonstrou potencial significativo para reduzir o consumo de papel na instituição. A digitalização de processos como diários de classe, fichas de matrícula e relatórios acadêmicos representa não apenas economia de recursos financeiros, mas também importante contribuição para a sustentabilidade ambiental.

Quanto ao projeto de Redes de Computadores, a elaboração do diagrama de rede no Cisco Packet Tracer permitiu visualizar e planejar adequadamente a infraestrutura necessária para a implantação do sistema, considerando aspectos como topologia, endereçamento IP e dimensionamento de equipamentos.

O sistema desenvolvido atende plenamente aos objetivos propostos, oferecendo uma solução integrada para a gestão acadêmica que combina eficiência operacional, inovação tecnológica e responsabilidade ambiental. A experiência prática adquirida neste projeto reforçou a importância do trabalho multidisciplinar e da aplicação integrada dos conhecimentos para o desenvolvimento de soluções tecnológicas completas e eficazes.

Como trabalhos futuros, sugere-se a implementação de interface web, integração com sistemas de gestão educacional existentes no mercado, e a incorporação de mais recursos de IA para análise preditiva de desempenho estudantil e recomendação personalizada de atividades.

- **REFERÊNCIAS**

A3A ENGENHARIA. Projeto de rede: guia de implementação de redes. Disponível em: https://a3aengenharia.com.br/conteudo/artigos-tecnicos/projeto-de-rede-guia-de-implementacao-de-redes/. Acesso em: 8 nov. 2025.

ALEXÔNICA. Os 7 componentes de uma rede: tudo o que você precisa saber. Disponível em: https://alexonica.com.br/os-7-componentes-de-uma-rede-tudo-o-que-voce-precisa-saber/. Acesso em: 8 nov. 2025.

XP EDUCAÇÃO. Topologias de rede. Disponível em: https://blog.xpeducacao.com.br/topologias-de-rede/. Acesso em: 8 nov. 2025.

SOFTDESIGN. Requisitos de software: funcionais e não funcionais. Disponível em:

https://softdesign.com.br/blog/requisitos-de-software-funcionais-e-nao-funcionais/. Acesso em: 8 nov. 2025.

DEVMEDIA. Ciclos de vida do software. Disponível em: https://www.devmedia.com.br/ciclos-de-vida-do-software/21099. Acesso em: 8 nov. 2025.

LUCIDCHART. O que é UML. Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-uml. Acesso em: 8 nov. 2025.

LUCIDCHART. Diagrama de caso de uso UML. Disponível em: https://www.lucidchart.com/pages/pt/diagrama-de-caso-de-uso-uml. Acesso em: 8 nov. 2025.

STOQUE. Melhores softwares educacionais. Disponível em: https://stoque.com.br/abaris/melhores-softwares-educacionais/. Acesso em: 8 nov. 2025.

CAPES. Manual técnico de avaliação de software de gestão escolar. Disponível em: https://educapes.capes.gov.br/bitstream/capes/572642/2/MANUAL%20T%C3%89CNICO%20DE%20AVALIA%C3%87%C3%83O%20DE%20SOFTWARE%20DE%20GEST%C3%83O%20ESCOLAR.pdf. Acesso em: 8 nov. 2025.

SCIELO. Sustentabilidade e o uso de papel na empresa. Disponível em: https://www.scielo.br/j/edur/a/vnXmgcZYr4hd9fW7ZSFGqMq/?format=html&lang=pt. Acesso em: 8 nov. 2025.

ARQUIVAR. Sustentabilidade: por que reduzir o uso de papel na empresa. Disponível em: https://arquivar.com.br/blog/sustentabilidade-por-que-reduzir-o-uso-de-papel-na-empresa/. Acesso em: 8 nov. 2025.

- **APÊNDICE A – CÓDIGO-FONTE DO SOFTWARE**

O código segue uma estrutura de arquivos onde se interagem entre si, armazenando e consultando informações de ambos.
Temos:

Sistema cadastro arquivos:

-login.cpp
-cJSON.h
-cJSON.c
-Usuario_School.json

Sistema de funções do site:

-homepage.cpp
-funcoes.cpp
-alunos.txt
-turmas.txt
-materias.txt
-relatorio.txt
-notas.txt
-fila.txt

Código de cadastro

```
#include <iostream> / / para copiar o código sem fonte para teste, vá até o final do projeto
#include <fstream>
#include <string>
#include <limits>
#include "cJSON.h"

#define ARQUIVO_USUARIOS "Usuario_School.json"

using namespace std;
```

```
struct Usuario {
   string nome;
   string senha;
};

bool login_user();
void cadastro_user();



int main() {
   int resp;
   bool login_sucesso = false;

   do {
      cout << "\n\nOla, bem-vindo a IETEP\n";
      cout << "Para prosseguir, insira seu login ou cadastre-se.\n\n";
      cout << "1 - Fazer LOGIN\n";
      cout << "2 - Cadastre-se\n\n";
      cout << "Escolha uma opcao: ";

      if (!(cin >> resp)) {
         cerr << "Entrada invalida. Encerrando." << endl;
         cin.clear();
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         continue;
      }

      switch (resp) {
         case 1:
            login_sucesso = login_user();
            break;
         case 2:
            cadastro_user();
            break;
```

```
            default:
                cout << "Opcao invalida. Tente novamente." << endl;
                break;
        }

    } while (!login_sucesso);

    cout << "\n--- Sessao de Usuario Iniciada ---\n" << endl;

    return 0;
}



bool login_user() {
    string nome_usuario, senha_usuario;

    cout << "\n\n-- Login --\n";
    cout << "Insira seu nome de usuario: ";
    cin >> nome_usuario;

    cout << "Insira sua senha: ";
    cin >> senha_usuario;

    ifstream arquivo(ARQUIVO_USUARIOS);
    if (!arquivo.is_open()) {
        cout << "Nenhum usuario cadastrado. Por favor, cadastre-se primeiro.\n";
        return false;
    }

    string json_string((istreambuf_iterator<char>(arquivo)), istreambuf_iterator<char>());
    arquivo.close();

    cJSON *root = cJSON_Parse(json_string.c_str());
```

```
    if (root == nullptr || !cJSON_IsArray(root)) {
        cout << "Formato de arquivo de usuarios invalido ou vazio.\n";
        cJSON_Delete(root);
        return false;
    }

    bool usuario_encontrado = false;
    int num_usuarios = cJSON_GetArraySize(root);

    for (int i = 0; i < num_usuarios; i++) {
        cJSON *usuario_json = cJSON_GetArrayItem(root, i);
        cJSON *j_nome = cJSON_GetObjectItem(usuario_json, "nome");
        cJSON *j_senha = cJSON_GetObjectItem(usuario_json, "senha");

        if (j_nome && j_senha && j_nome->valuestring && j_senha->valuestring) {
            if (nome_usuario == j_nome->valuestring && senha_usuario ==
j_senha->valuestring) {
                usuario_encontrado = true;
                cout << "Login bem-sucedido! Bem-vindo, " << j_nome->valuestring << ".\n";
                break;
            }
        }
    }

    if (!usuario_encontrado) {
        cout << "Nome de usuario ou senha incorretos. Tente novamente.\n";
    }

    cJSON_Delete(root);
    return usuario_encontrado;
}

void cadastro_user() {
    Usuario novo_usuario;
    string confirma_senha;
```

```
    cout << "\n\n-- Cadastro --\n";
    cout << "Insira seu nome de usuario: ";
    cin >> novo_usuario.nome;

    do {
        cout << "Insira a sua senha: ";
        cin >> novo_usuario.senha;

        cout << "Confirme a sua senha: ";
        cin >> confirma_senha;

        if (novo_usuario.senha != confirma_senha) {
            cout << "\nERRO: As senhas nao coincidem. Tente novamente.\n\n";
        }
    } while (novo_usuario.senha != confirma_senha);

    cout << "\nSenhas confirmadas. Prosseguindo...\n";

    cJSON *root = nullptr;
    ifstream arquivo_leitura(ARQUIVO_USUARIOS);

    if (arquivo_leitura.is_open()) {
        string json_string((istreambuf_iterator<char>(arquivo_leitura)),
istreambuf_iterator<char>());
        arquivo_leitura.close();
        if (!json_string.empty()) {
            root = cJSON_Parse(json_string.c_str());
        }
    }

    if (root == nullptr || !cJSON_IsArray(root)) {
        cJSON_Delete(root);
        root = cJSON_CreateArray();
    }

    cJSON *usuario_json = cJSON_CreateObject();
    cJSON_AddStringToObject(usuario_json, "nome", novo_usuario.nome.c_str());
    cJSON_AddStringToObject(usuario_json, "senha", novo_usuario.senha.c_str());
    cJSON_AddItemToArray(root, usuario_json);
```

```
    char *json_para_salvar = cJSON_Print(root);

    ofstream arquivo_escrita(ARQUIVO_USUARIOS);
    if (arquivo_escrita.is_open()) {
        arquivo_escrita << json_para_salvar;
        arquivo_escrita.close();
        cout << "Usuario cadastrado com sucesso!\n";
    } else {
        cout << "ERRO CRITICO: Nao foi possivel salvar o novo usuario!\n";
    }

    cJSON_Delete(root);
    free(json_para_salvar);
}
```

Código de Funções

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <limits>
#include <cstddef>
#include <sstream>
#include <iomanip>

using namespace std;

struct Aluno {
    string nome;
    int matricula;
    string curso;
```

```
    int faltas;
};

struct Materia {
    string codigo;
    string nome;
};

// Funções Auxiliares
void limpar_buffer();
vector<Aluno> ler_alunos_do_arquivo();
void salvar_alunos_no_arquivo(const vector<Aluno>& alunos);
vector<Materia> ler_materias_do_arquivo();
vector<int> ler_fila_do_arquivo();
void salvar_fila_no_arquivo(const vector<int>& fila);

// Funções do Menu Principal
void matricular_aluno();
void matricular_turma();
void ver_turmas();
void aplicar_falta();
void ver_frequencia();
void lancar_notas();
void ver_notas();
void relatorio();
void fila_atendimento();
void chamar_proximo_da_fila();

int main() {
    int opcao;

    do {
        cout << "\n\n=== Sistema de Gestao Academica ===\n";
        cout << "1  - Matricular Aluno\n";
        cout << "2  - Matricular Turma\n";
        cout << "3  - Ver Turmas\n";
        cout << "4  - Aplicar Falta\n";
```

```
        cout << "5  - Ver Frequencia\n";
        cout << "6  - Lancar Nota\n";
        cout << "7  - Ver Notas\n";
        cout << "8  - Escrever Relatorio\n";
        cout << "9  - Fila de Atendimento\n";
        cout << "10 - Chamar Proximo da Fila\n";
        cout << "0  - Sair\n";
        cout << "-------------------------------------\n";
        cout << "Escolha uma opcao: ";

        if (!(cin >> opcao)) {
            cout << "\nOpcao invalida! Por favor, digite um numero.\n";
            limpar_buffer();
            continue;
        }

        limpar_buffer();

        switch (opcao) {
            case 1: matricular_aluno(); break;
            case 2: matricular_turma(); break;
            case 3: ver_turmas(); break;
            case 4: aplicar_falta(); break;
            case 5: ver_frequencia(); break;
            case 6: lancar_notas(); break;
            case 7: ver_notas(); break;
            case 8: relatorio(); break;
            case 9: fila_atendimento(); break;
            case 10: chamar_proximo_da_fila(); break;
            case 0: cout << "Saindo do sistema...\n"; break;
            default: cout << "Opcao invalida! Tente novamente.\n";
        }
    } while (opcao != 0);

    return 0;
}

void limpar_buffer() {
```

```
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

vector<Aluno> ler_alunos_do_arquivo() {
    vector<Aluno> alunos;
    ifstream arquivo("alunos.txt");
    if (arquivo.is_open()) {
        string linha;
        while (getline(arquivo, linha)) {
            stringstream ss(linha);
            string campo;
            Aluno aluno;
            if (getline(ss, aluno.nome, ',') &&
                getline(ss, campo, ',') &&
                getline(ss, aluno.curso, ',') &&
                getline(ss, campo, ',')) {
                aluno.matricula = stoi(campo);
                aluno.faltas = stoi(campo); // Re-leitura do campo matricula aqui, corrigir

                 // Correção na leitura de dados
                stringstream ss_corrigido(linha);
                getline(ss_corrigido, aluno.nome, ',');
                getline(ss_corrigido, campo, ',');
                aluno.matricula = stoi(campo);
                getline(ss_corrigido, aluno.curso, ',');
                getline(ss_corrigido, campo, ',');
                aluno.faltas = stoi(campo);

                alunos.push_back(aluno);
            }
        }
        arquivo.close();
    }
    return alunos;
}
```

```
void salvar_alunos_no_arquivo(const vector<Aluno>& alunos) {
    ofstream arquivo("alunos.txt");
    if (arquivo.is_open()) {
        for (const Aluno& aluno : alunos) {
            arquivo << aluno.nome << ","
                    << aluno.matricula << ","
                    << aluno.curso << ","
                    << aluno.faltas << endl;
        }
        arquivo.close();
    } else {
        cout << "ERRO CRITICO: Nao foi possivel abrir alunos.txt para salvar!\n";
    }
}

vector<Materia> ler_materias_do_arquivo() {
    vector<Materia> materias;
    ifstream arquivo("materias.txt");
    if (arquivo.is_open()) {
        string linha;
        while (getline(arquivo, linha)) {
            stringstream ss(linha);
            string codigo, nome;
            if (getline(ss, codigo, ',') && getline(ss, nome)) {
                materias.push_back({codigo, nome});
            }
        }
        arquivo.close();
    }
    return materias;
}

vector<int> ler_fila_do_arquivo() {
    vector<int> fila;
    ifstream arquivo("fila.txt");
    if (arquivo.is_open()) {
```

```
        int matricula;
        while (arquivo >> matricula) {
            fila.push_back(matricula);
        }
        arquivo.close();
    }
    return fila;
}

void salvar_fila_no_arquivo(const vector<int>& fila) {
    ofstream arquivo("fila.txt");
    if (arquivo.is_open()) {
        for (int matricula : fila) {
            arquivo << matricula << endl;
        }
        arquivo.close();
    } else {
        cout << "ERRO CRITICO: Nao foi possivel salvar o estado da fila!\n";
    }
}



void matricular_aluno() {
    cout << "\n--------- Matricular Aluno ---------\n";
    Aluno aluno;

    cout << "Digite o nome do aluno: ";
    getline(cin, aluno.nome);

    cout << "Digite o numero de matricula: ";
    cin >> aluno.matricula;
    limpar_buffer();

    cout << "Digite o curso: ";
    getline(cin, aluno.curso);

    cout << "Digite o numero inicial de faltas (0 se nao tiver): ";
```

```
    cin >> aluno.faltas;
    limpar_buffer();

    ofstream arquivo("alunos.txt", ios::app);
    if (arquivo.is_open()) {
        arquivo << aluno.nome << "," << aluno.matricula << "," << aluno.curso << "," <<
aluno.faltas << endl;
        arquivo.close();
        cout << "Aluno matriculado com sucesso!\n";
    } else {
        cout << "Erro ao abrir o arquivo para gravacao!\n";
    }
}

void matricular_turma() {
    cout << "\n--------- Matricular Turma ---------\n";
    string nome_turma;
    cout << "Digite o nome da nova turma: ";
    getline(cin, nome_turma);

    vector<Aluno> todos_alunos = ler_alunos_do_arquivo();
    if (todos_alunos.empty()) {
        cout << "Nenhum aluno cadastrado no sistema.\n";
        return;
    }

    vector<int> matriculas_na_turma;
    int numero_aluno;

    while (true) {
        cout << "\n--- Alunos Disponiveis ---\n";
        for (size_t i = 0; i < todos_alunos.size(); ++i) {
            cout << i + 1 << " - " << todos_alunos[i].nome << "\n";
        }
        cout << "0 - Finalizar e Salvar Turma\n";
        cout << "Digite o numero do aluno para adicionar: ";
```

```
        cin >> numero_aluno;
        if (cin.fail() || numero_aluno < 0 || numero_aluno > todos_alunos.size()) {
            cout << "Entrada invalida!\n";
            limpar_buffer();
            continue;
        }

        if (numero_aluno == 0) break;

        matriculas_na_turma.push_back(todos_alunos[numero_aluno - 1].matricula);
        cout << "Aluno '" << todos_alunos[numero_aluno - 1].nome << "' adicionado.\n";
        todos_alunos.erase(todos_alunos.begin() + (numero_aluno - 1));
    }

    limpar_buffer();

    if (matriculas_na_turma.empty()) {
        cout << "Nenhum aluno adicionado. Turma nao criada.\n";
        return;
    }

    ofstream arquivo("turmas.txt", ios::app);
    if (arquivo.is_open()) {
        arquivo << nome_turma << ":";
        for (size_t i = 0; i < matriculas_na_turma.size(); ++i) {
            arquivo << matriculas_na_turma[i] << (i == matriculas_na_turma.size() - 1 ? "" : ",");
        }
        arquivo << endl;
        arquivo.close();
        cout << "Turma '" << nome_turma << "' salva com sucesso!\n";
    } else {
        cout << "Erro ao abrir o arquivo turmas.txt!\n";
    }
}
```

```
void ver_turmas() {
    cout << "\n--------- Turmas Cadastradas ---------\n";
    vector<Aluno> todos_alunos = ler_alunos_do_arquivo();
    ifstream arquivo_turmas("turmas.txt");

    if (!arquivo_turmas.is_open() || arquivo_turmas.peek() == EOF) {
        cout << "Nenhuma turma foi criada ainda.\n";
        return;
    }

    string linha_turma;
    while (getline(arquivo_turmas, linha_turma)) {
        stringstream ss_linha(linha_turma);
        string nome_turma, matriculas_str;
        getline(ss_linha, nome_turma, ':');
        getline(ss_linha, matriculas_str);

        cout << "\n--- Turma: " << nome_turma << " ---\n";
        stringstream ss_matriculas(matriculas_str);
        string matricula_individual_str;

        while (getline(ss_matriculas, matricula_individual_str, ',')) {
            int matricula_aluno = stoi(matricula_individual_str);
            bool aluno_encontrado = false;
            for (const Aluno& aluno : todos_alunos) {
                if (aluno.matricula == matricula_aluno) {
                    cout << "  - Nome: " << aluno.nome << " | Matricula: " << aluno.matricula <<
"\n";
                    aluno_encontrado = true;
                    break;
                }
            }
            if (!aluno_encontrado) {
                cout << "  - Aluno com matricula " << matricula_aluno << " nao encontrado.\n";
            }
        }
```

```
    }
    arquivo_turmas.close();
}


void aplicar_falta() {
    cout << "\n--------- Aplicar Falta ---------\n";
    vector<Aluno> alunos = ler_alunos_do_arquivo();
    if (alunos.empty()) {
        cout << "Nenhum aluno cadastrado no sistema.\n";
        return;
    }


    cout << "Selecione o aluno para aplicar a falta:\n";
    for (size_t i = 0; i < alunos.size(); ++i) {
        cout << i + 1 << " - " << alunos[i].nome << " (Faltas atuais: " << alunos[i].faltas <<
")\n";
    }
    cout << "0 - Voltar ao menu\n";
    cout << "Escolha uma opcao: ";


    int escolha;
    cin >> escolha;
    if (cin.fail() || escolha < 0 || escolha > alunos.size()) {
        cout << "Opcao invalida!\n";
        limpar_buffer();
        return;
    }
    if (escolha == 0) {
        limpar_buffer();
        return;
    }


    int indice_aluno = escolha - 1;
    int faltas_a_adicionar;
    cout << "Digite o numero de faltas a ADICIONAR para " << alunos[indice_aluno].nome
<< ": ";
    cin >> faltas_a_adicionar;
    if (cin.fail() || faltas_a_adicionar < 0) {
```

```
        cout << "Numero de faltas invalido!\n";
        limpar_buffer();
        return;
    }

    alunos[indice_aluno].faltas += faltas_a_adicionar;
    salvar_alunos_no_arquivo(alunos);

    cout << "Faltas aplicadas com sucesso!\n";
    cout << "Novo total de faltas de " << alunos[indice_aluno].nome << ": " <<
alunos[indice_aluno].faltas << "\n";
    limpar_buffer();
}

void ver_frequencia() {
    cout << "\n--------- Ver Frequencia ---------\n";
    vector<Aluno> alunos = ler_alunos_do_arquivo();
    if (alunos.empty()) {
        cout << "Nenhum aluno cadastrado para verificar a frequencia.\n";
        return;
    }

    int total_aulas;
    cout << "Digite o numero total de aulas do curso/semestre: ";
    cin >> total_aulas;
    if (cin.fail() || total_aulas <= 0) {
        cout << "Numero total de aulas invalido!\n";
        limpar_buffer();
        return;
    }

    cout << "\n--- Relatorio de Frequencia ---\n";
    cout << fixed << setprecision(2);

    for (const Aluno& aluno : alunos) {
        double porcentagem_faltas = ((double)aluno.faltas / total_aulas) * 100.0;
        double porcentagem_presenca = 100.0 - porcentagem_faltas;
```

```
        if (porcentagem_presenca < 0) {
            porcentagem_presenca = 0;
        }

        cout << "Aluno: " << aluno.nome
             << " | Faltas: " << aluno.faltas
             << " | Frequencia: " << porcentagem_presenca << "%\n";
    }
    cout << "--------------------------------\n";
    limpar_buffer();
}

void lancar_notas() {
    cout << "\n--------- Lancar Nota ---------\n";
    vector<Aluno> alunos = ler_alunos_do_arquivo();
    vector<Materia> materias = ler_materias_do_arquivo();

    if (alunos.empty()) {
        cout << "Nenhum aluno cadastrado no sistema.\n";
        return;
    }
    if (materias.empty()) {
        cout << "AVISO: Nenhuma materia cadastrada em 'materias.txt'.\n";
        cout << "Crie o arquivo 'materias.txt' com o conteudo 'CODIGO,NOME' por linha.\n";
        return;
    }

    cout << "Selecione o aluno:\n";
    for (size_t i = 0; i < alunos.size(); ++i) {
        cout << i + 1 << " - " << alunos[i].nome << "\n";
    }
    cout << "0 - Voltar\n";
    cout << "Escolha uma opcao: ";
    int escolha_aluno;
    cin >> escolha_aluno;
    if (cin.fail() || escolha_aluno < 0 || escolha_aluno > alunos.size()) {
```

```
    cout << "Opcao invalida!\n";
    limpar_buffer();
    return;
}
if (escolha_aluno == 0) {
    limpar_buffer();
    return;
}
Aluno aluno_selecionado = alunos[escolha_aluno - 1];


cout << "\n--- Materias Disponiveis ---\n";
for(const auto& materia : materias) {
    cout << "Codigo: " << materia.codigo << " - Nome: " << materia.nome << "\n";
}
cout << "----------------------------\n";
cout << "Digite o CODIGO da materia: ";
string codigo_materia;
cin >> codigo_materia;

bool codigo_valido = false;
for(const auto& materia : materias) {
    if (materia.codigo == codigo_materia) {
        codigo_valido = true;
        break;
    }
}
if (!codigo_valido) {
    cout << "Codigo de materia invalido!\n";
    limpar_buffer();
    return;
}


double nota;
cout << "Digite a nota para " << aluno_selecionado.nome << ": ";
cin >> nota;
if (cin.fail() || nota < 0 || nota > 10) {
    cout << "Nota invalida! Deve ser entre 0 e 10.\n";
    limpar_buffer();
```

```
        return;
    }

    ofstream arquivo("notas.txt", ios::app);
    if (arquivo.is_open()) {
        arquivo << aluno_selecionado.matricula << "," << codigo_materia << "," << nota <<
endl;
        arquivo.close();
        cout << "Nota lancada com sucesso!\n";
    } else {
        cout << "Erro ao abrir o arquivo notas.txt!\n";
    }
    limpar_buffer();
}

void ver_notas() {
    cout << "\n--------- Boletim de Notas ---------\n";
    vector<Aluno> alunos = ler_alunos_do_arquivo();
    vector<Materia> materias = ler_materias_do_arquivo();
    ifstream arquivo_notas("notas.txt");

    if (alunos.empty()) {
        cout << "Nenhum aluno cadastrado.\n";
        return;
    }
    if (!arquivo_notas.is_open() || arquivo_notas.peek() == EOF) {
        cout << "Nenhuma nota foi lancada ainda.\n";
        return;
    }

    for (const auto& aluno : alunos) {
        cout << "\n--- Aluno: " << aluno.nome << " (Matricula: " << aluno.matricula << ") ---\n";

        arquivo_notas.clear();
        arquivo_notas.seekg(0, ios::beg);

        string linha;
```

```
        bool encontrou_nota = false;
        while (getline(arquivo_notas, linha)) {
            stringstream ss(linha);
            string matricula_str, codigo_materia, nota_str;
            getline(ss, matricula_str, ',');
            getline(ss, codigo_materia, ',');
            getline(ss, nota_str, ',');

            if (stoi(matricula_str) == aluno.matricula) {
                encontrou_nota = true;
                string nome_materia = "Materia Desconhecida";
                for (const auto& materia : materias) {
                    if (materia.codigo == codigo_materia) {
                        nome_materia = materia.nome;
                        break;
                    }
                }
                cout << "  - Materia: " << nome_materia << " | Nota: " << stod(nota_str) << "\n";
            }
        }

        if (!encontrou_nota) {
            cout << "  (Nenhuma nota lancada para este aluno)\n";
        }
    }
    arquivo_notas.close();
}

void relatorio() {
    cout << "\n--------- Escrever Relatorio ---------\n";
    int matricula;
    cout << "Para identificar, digite sua matricula: ";
    if (!(cin >> matricula)) {
        cout << "Matricula invalida! Apenas numeros.\n";
        limpar_buffer();
        return;
    }
```

```
    limpar_buffer();

    string texto_relatorio;
    cout << "Digite seu relatorio (pressione ENTER para salvar):\n> ";
    getline(cin, texto_relatorio);

    if (texto_relatorio.empty()) {
        cout << "Relatorio vazio. Nada foi salvo.\n";
        return;
    }

    ofstream arquivo("relatorio.txt", ios::app);
    if (arquivo.is_open()) {
        arquivo << "Matricula " << matricula << ": " << texto_relatorio << endl;
        arquivo.close();
        cout << "Relatorio salvo com sucesso!\n";
    } else {
        cout << "Erro ao abrir o arquivo relatorio.txt!\n";
    }
}

void fila_atendimento() {
    cout << "\n--------- Fila de Atendimento da Secretaria ---------\n";
    vector<int> fila = ler_fila_do_arquivo();
    vector<Aluno> todos_alunos = ler_alunos_do_arquivo();

    if (fila.empty()) {
        cout << "A fila de atendimento esta vazia.\n";
    } else {
        cout << "Pessoas na fila no momento:\n";
        for (size_t i = 0; i < fila.size(); ++i) {
            string nome_aluno = "Desconhecido";
            for (const auto& aluno : todos_alunos) {
                if (aluno.matricula == fila[i]) {
                    nome_aluno = aluno.nome;
                    break;
                }
```

```
        }
        cout << i + 1 << "o - " << nome_aluno << " (Matricula: " << fila[i] << ")\n";
    }
}

cout << "\nDigite sua matricula para entrar na fila (ou 0 para voltar): ";
int nova_matricula;
cin >> nova_matricula;

if (cin.fail()) {
    cout << "Entrada invalida.\n";
    limpar_buffer();
    return;
}
if (nova_matricula == 0) {
    limpar_buffer();
    return;
}

bool aluno_existe = false;
for (const auto& aluno : todos_alunos) {
    if (aluno.matricula == nova_matricula) {
        aluno_existe = true;
        break;
    }
}
if (!aluno_existe) {
    cout << "Erro: Aluno com matricula " << nova_matricula << " nao encontrado.\n";
    limpar_buffer();
    return;
}

for (int matricula_na_fila : fila) {
    if (matricula_na_fila == nova_matricula) {
        cout << "Voce ja esta na fila!\n";
        limpar_buffer();
        return;
    }
```

```
    }

    fila.push_back(nova_matricula);
    salvar_fila_no_arquivo(fila);
    cout << "Voce foi adicionado a fila! Sua posicao e: " << fila.size() << "o\n";
    limpar_buffer();
}

void chamar_proximo_da_fila() {
    cout << "\n--------- Chamar Proximo da Fila ---------\n";
    vector<int> fila = ler_fila_do_arquivo();

    if (fila.empty()) {
        cout << "Nao ha ninguem na fila para chamar.\n";
        return;
    }

    int matricula_atendida = fila.front();
    vector<Aluno> todos_alunos = ler_alunos_do_arquivo();
    string nome_aluno = "Desconhecido";
    for (const auto& aluno : todos_alunos) {
        if (aluno.matricula == matricula_atendida) {
            nome_aluno = aluno.nome;
            break;
        }
    }

    cout << "Chamando para atendimento: " << nome_aluno << " (Matricula: " <<
matricula_atendida << ")\n";

    fila.erase(fila.begin());
    salvar_fila_no_arquivo(fila);

    cout << "Aluno atendido e removido da fila.\n";
}
```

ANÁLISE DO SISTEMA DE CADASTRO (login.cpp)

O arquivo login.cpp constitui o módulo de autenticação do sistema, sendo a porta de entrada para os usuários. Sua principal responsabilidade é gerenciar o acesso, oferecendo as opções de login para usuários já existentes e de cadastro para novos usuários. As credenciais, compostas por nome de usuário e senha, são armazenadas de forma estruturada no arquivo Usuario_School.json. A seguir, detalha-se o funcionamento de seus componentes principais.

1.1 Estrutura e Inicialização do Programa

O programa inicia com a inclusão das bibliotecas essenciais para seu funcionamento: iostream para operações de entrada e saída de dados no console, fstream para a manipulação de arquivos, string para o trabalho com textos, e a biblioteca externa cJSON.h, que é fundamental para a leitura e escrita no arquivo de usuários formatado em JSON. Adicionalmente, é definida a estrutura Usuario, um molde simples para organizar o nome e a senha de cada usuário em memória.

A função main atua como o núcleo operacional deste módulo. Ao ser executada, ela apresenta ao usuário uma tela de boas-vindas e um menu interativo com duas alternativas: "Fazer LOGIN" e "Cadastre-se". O sistema então entra em um ciclo de repetição (do-while) que persiste até que um login seja realizado com sucesso. Com base na opção selecionada pelo usuário, o programa direciona o fluxo para a função correspondente, login_user() ou cadastro_user(). Uma vez que o login é validado, uma mensagem de confirmação é exibida e a execução deste módulo é concluída, permitindo que o usuário avance para as demais funcionalidades do sistema.

1.2 Função de Autenticação de Usuário (login_user)

Esta função é projetada para verificar a validade das credenciais fornecidas por um usuário. O processo se inicia com a solicitação do nome de usuário e da senha. Em seguida, o sistema tenta acessar o arquivo Usuario_School.json, que funciona como a base de dados de usuários. Caso o arquivo não seja encontrado, o programa informa que não há usuários cadastrados e retorna ao menu principal.

Se o arquivo for localizado, seu conteúdo é lido e interpretado pela biblioteca cJSON, que converte o texto em uma estrutura de dados manipulável. A função então percorre a lista de usuários armazenados, comparando, para cada um, o nome e a senha com as informações que foram inseridas pelo usuário. Se uma correspondência exata for encontrada, uma mensagem de boas-vindas é exibida, e a função sinaliza o sucesso da operação. Caso contrário, se a lista for percorrida por completo sem encontrar uma combinação válida, o sistema informa que os dados estão incorretos e solicita uma nova tentativa.

1.3 Função de Cadastro de Novo Usuário (cadastro_user)

Quando um indivíduo opta por se cadastrar, esta função é acionada. Ela orienta o novo usuário no processo de criação de uma conta. Primeiramente, solicita a definição de um nome de usuário e de uma senha. Para garantir a segurança e evitar erros de digitação, é implementado

um mecanismo de confirmação, no qual a senha deve ser inserida duas vezes. Um laço de verificação assegura que o processo só avance quando ambas as inserções forem idênticas.

Com as credenciais confirmadas, a função se prepara para salvar os novos dados. Ela abre o arquivo Usuario_School.json e carrega a lista de usuários já existentes. Essa etapa é crucial para garantir que os registros anteriores não sejam perdidos. Se o arquivo estiver vazio ou não existir, uma nova estrutura de lista (array JSON) é criada. A partir daí, o novo usuário seria adicionado a essa lista, que, por sua vez, seria salva novamente no arquivo, concluindo o registro.

2 ANÁLISE DO SISTEMA DE GESTÃO ACADÊMICA (funcoes.cpp)

Mergulhando no coração do sistema, encontramos o arquivo que orquestra todas as operações do dia a dia acadêmico. Sua arquitetura foi pensada para ser amigável, centrada em um menu interativo que convida o usuário a explorar suas diversas capacidades, desde a gestão de alunos até o atendimento na secretaria. Para que nenhuma informação se perca, o sistema confia em uma série de arquivos de texto (.txt), que funcionam como sua memória permanente, guardando cuidadosamente cada detalhe sobre alunos, turmas, matérias e todos os outros registros vitais.

2.1 A Organização dos Dados e sua Persistência

Para que o sistema possa compreender e manipular as informações de forma coerente, foram definidos modelos de dados que refletem o mundo real da instituição. São eles:

- **A figura do Aluno**: Representada pela estrutura Aluno, que guarda não apenas o nome e a matrícula, mas também o curso e o registro de sua frequência, informações essenciais para sua jornada acadêmica.
- **A identidade da Matéria**: Representada pela estrutura Materia, que dá a cada disciplina um código único e um nome, facilitando sua identificação em todo o sistema.

Toda informação gerada, desde uma nova matrícula até o lançamento de uma nota, é diligentemente salva nestes arquivos de texto. Essa abordagem, embora simples, confere ao sistema uma memória duradoura, garantindo que cada registro permaneça seguro e acessível sempre que o programa for iniciado. Cada arquivo tem sua vocação: alunos.txt é o livro de registros dos estudantes, turmas.txt é o mapa das classes, e assim por diante, criando um ecossistema de dados organizado e funcional.

2.2 Os Bastidores: Funções de Apoio à Manipulação de Dados

Seguindo boas práticas de programação, o código adota uma estratégia de modularização, criando um conjunto de funções auxiliares dedicadas exclusivamente a conversar com os arquivos de texto. Funções como ler_alunos_do_arquivo() e salvar_alunos_no_arquivo() atuam como intermediários especializados: sua missão é buscar os dados nos arquivos, traduzi-los para um formato que o programa entenda e, ao final de uma operação, levar os dados atualizados de volta para seu local de armazenamento. Essa separação de tarefas torna o

código mais limpo e confiável, centralizando a responsabilidade de lidar com os arquivos em um único lugar.

2.3 O Painel de Controle: Interação e Funcionalidades

A função `main` serve como o centro de comando da aplicação, sendo o primeiro ponto de contato com o usuário. É ela que apresenta o painel de controle, um menu claro e objetivo com todas as ferramentas disponíveis. O sistema permanece à disposição do usuário, em um ciclo contínuo, pronto para atender a qualquer solicitação, até que o operador decida encerrar a sessão. As funcionalidades foram agrupadas de forma lógica para facilitar a navegação.

2.3.1 Gestão da Comunidade Acadêmica: Alunos e Turmas

Este módulo é dedicado à construção e organização do corpo discente.

- **Matricular Aluno**: O processo de acolhimento de um novo membro na comunidade acadêmica inicia-se aqui. A função guia o administrador no registro de todas as informações cruciais do estudante, que são imediatamente perpetuadas no arquivo `alunos.txt`.
- **Matricular Turma**: Mais do que apenas agrupar nomes, esta função permite a criação de verdadeiras classes. De forma intuitiva, o sistema mostra quem são os alunos disponíveis e permite que o usuário os adicione a uma nova turma, que é então oficialmente registrada no arquivo `turmas.txt`.
- **Ver Turmas**: Para uma visão panorâmica da organização das classes, esta ferramenta consulta os registros e apresenta um relatório claro de todas as turmas, listando nominalmente os alunos que compõem cada uma.

2.3.2 O Coração Pedagógico: Acompanhamento de Desempenho

Aqui se concentram as ferramentas essenciais para o acompanhamento do progresso dos alunos.

- **Aplicar Falta**: Para um controle de frequência preciso e justo, esta ferramenta permite registrar as ausências de um aluno específico, atualizando seu histórico instantaneamente.
- **Ver Frequência**: Com base nos registros de faltas, esta função oferece um panorama da assiduidade dos alunos, calculando e exibindo a porcentagem de presença de cada um, um indicador fundamental para o sucesso acadêmico.
- **Lançar Notas**: No centro do processo avaliativo, esta função oferece um caminho claro e seguro para o registro do desempenho acadêmico, vinculando uma nota a um aluno e a uma matéria específica.
- **Ver Notas**: Esta funcionalidade materializa o esforço dos alunos em um boletim completo, percorrendo os registros para exibir, de maneira organizada, as notas de cada estudante em todas as disciplinas cursadas.

2.3.3 Suporte Administrativo e Atendimento Humanizado

Este conjunto de ferramentas visa a otimizar a comunicação e os serviços de apoio.

- **Escrever Relatório**: Um canal aberto de comunicação, onde qualquer usuário identificado pode registrar observações, solicitações ou pareceres, que ficam devidamente armazenados para consulta futura.
- **Fila de Atendimento**: Visando organizar o fluxo de pessoas e otimizar o tempo de todos, o sistema implementa uma fila de atendimento virtual. Os alunos podem se registrar e acompanhar sua posição, trazendo ordem e transparência ao serviço da secretaria.
- **Chamar Próximo da Fila**: Para dar fluidez ao atendimento, esta função executa a ação de chamar a próxima pessoa da fila, anunciando seu nome e matrícula, e atualizando a fila para que o processo continue de forma eficiente e respeitosa.

Imagens do programa em funcionamento

CADASTRO

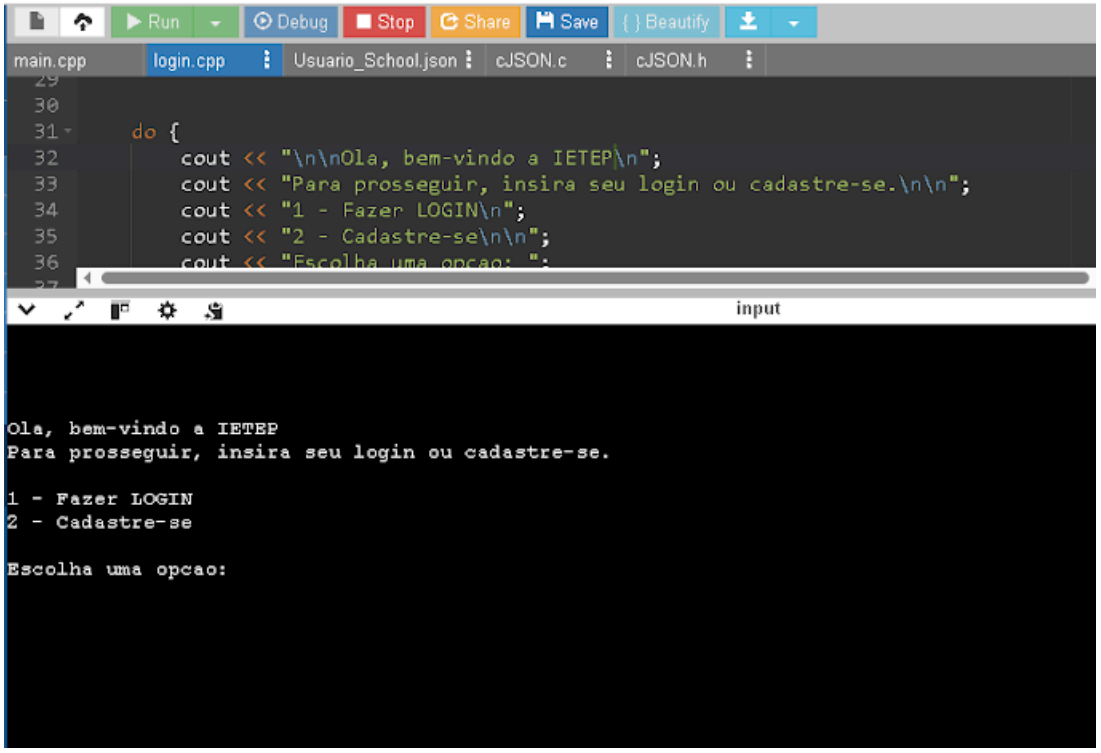


Imagem 1: Menu Inicial

O programa é executado e apresenta o menu principal no console. O usuário visualiza as opções disponíveis: "1" para realizar o login caso já tenha conta, ou "2" para cadastrar um novo usuário no sistema.

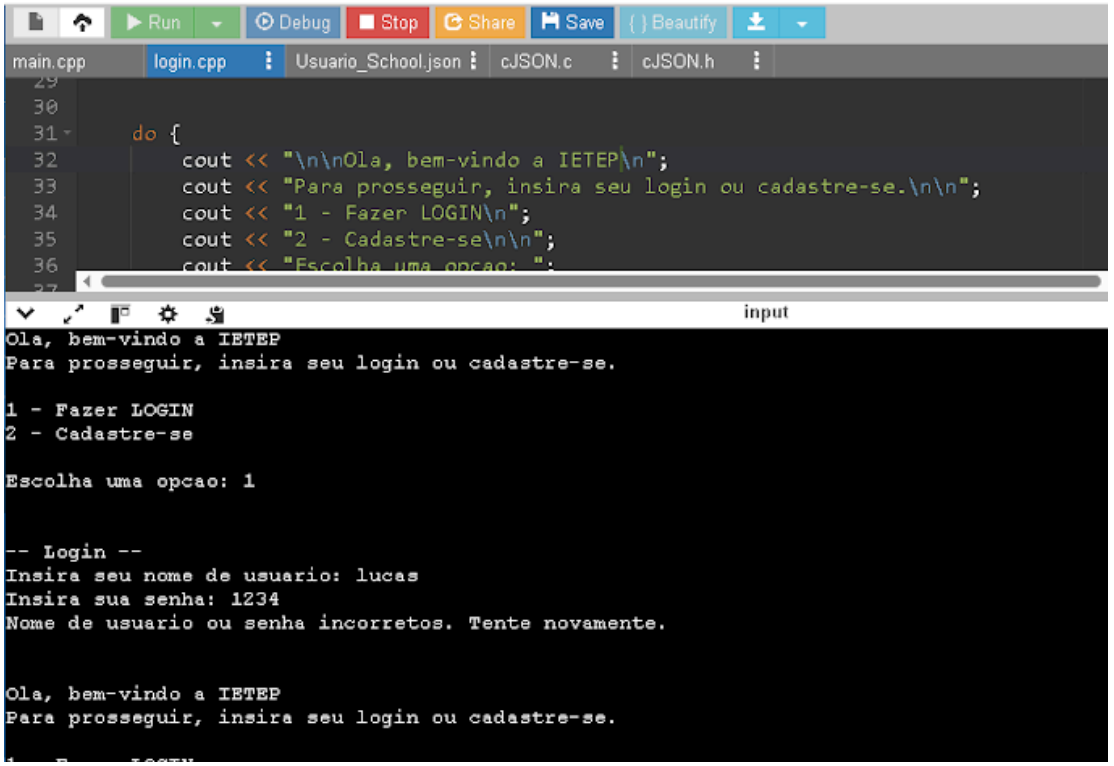


Imagem 2: Tentativa de Login Inválida

O usuário tenta fazer login com a conta "lucas", que ainda não existe. O sistema busca esse registro no arquivo JSON, não encontra correspondência e informa corretamente que o usuário ou senha estão incorretos.

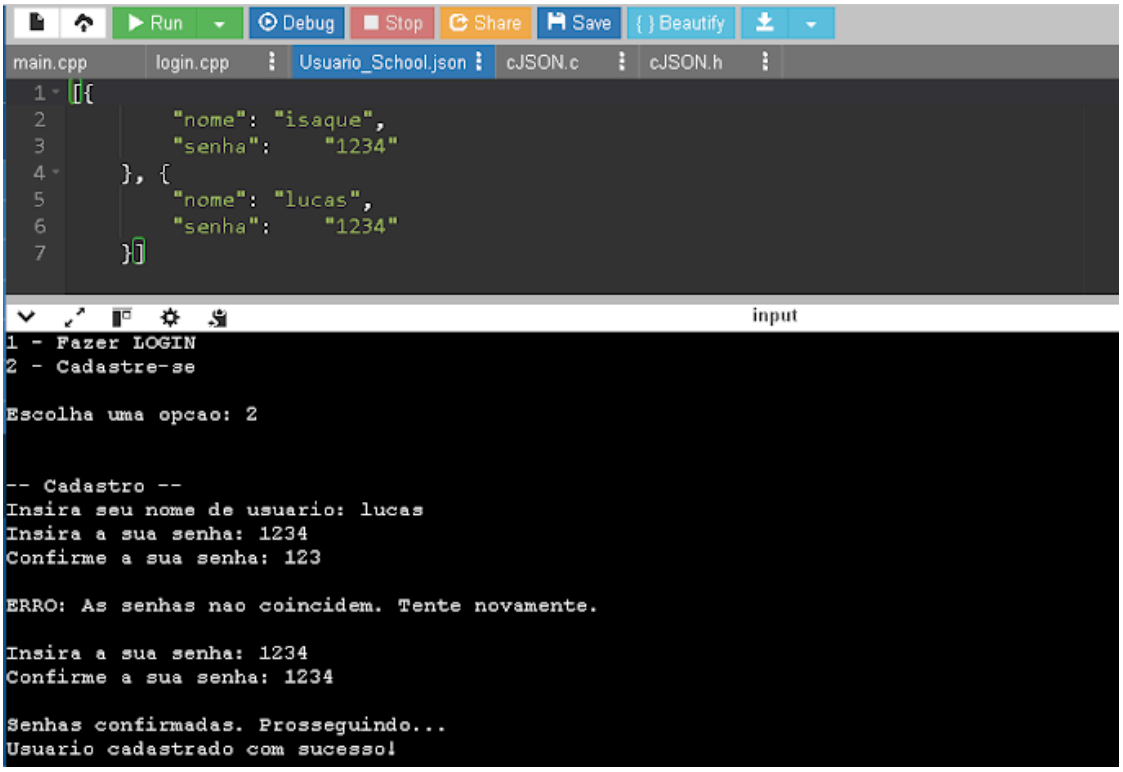


Imagem 3: Processo de Cadastro

O usuário seleciona a opção de cadastro (2) e cria a conta "lucas". O sistema verifica a

confirmação da senha (mostrando erro se não coincidirem) e, ao final, salva os novos dados permanentemente no arquivo Usuario_School.json.

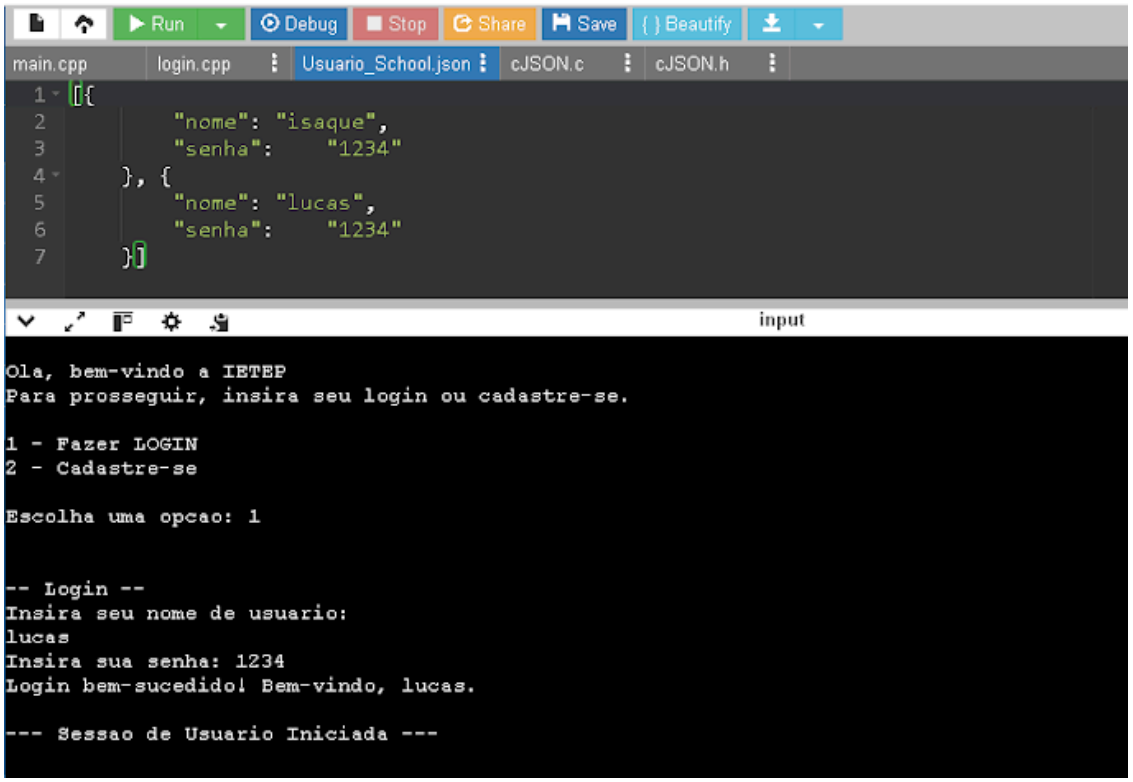


Imagem 4: Login Bem-Sucedido

Agora que o registro foi salvo no arquivo, o usuário seleciona a opção de login (1) novamente. O sistema valida as credenciais de "lucas" com sucesso e libera o acesso, iniciando a sessão.

FUNÇÕES

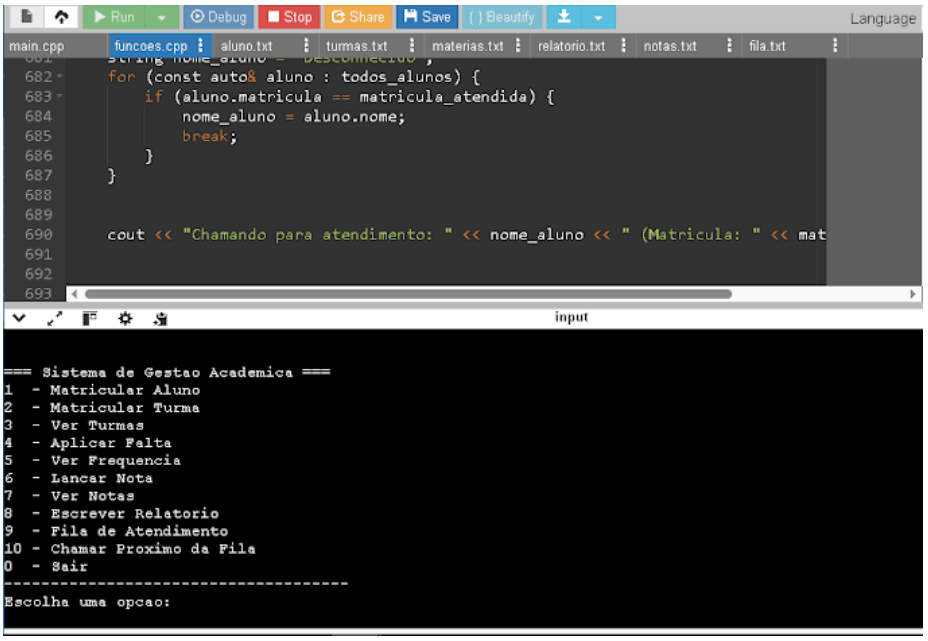


Imagem 1: Menu do Sistema de Gestão

Após o login, o programa principal é iniciado, exibindo o menu do Sistema de Gestão Acadêmica. O usuário agora tem acesso a todas as funcionalidades, como matricular alunos, gerenciar turmas, lançar notas e faltas, entre outras opções.

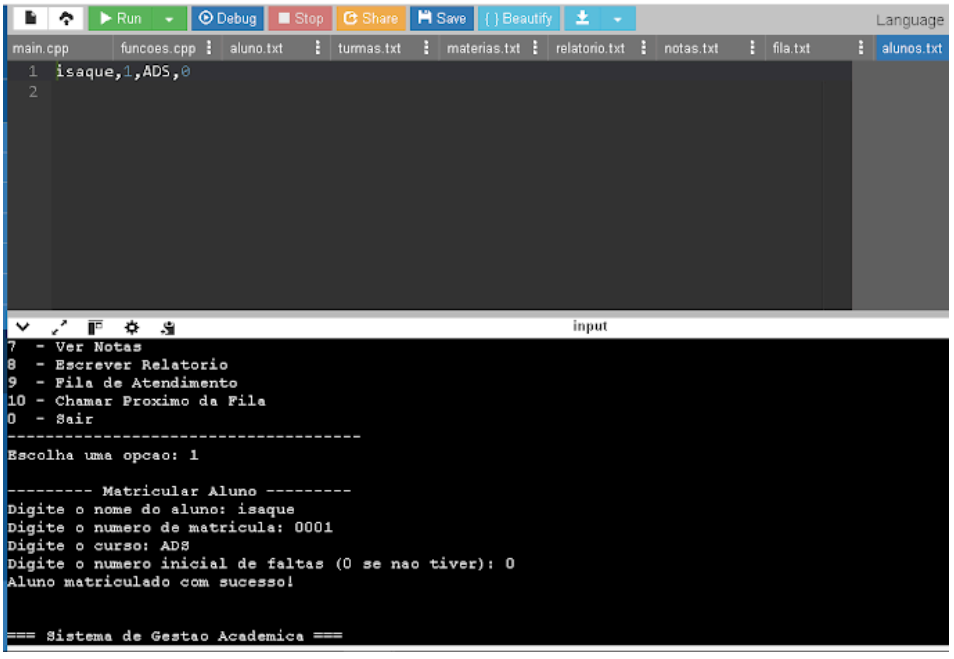


Imagem 2: Matricular Aluno (Opção 1)

O usuário seleciona a opção 1 para matricular um novo aluno no sistema. Ele insere os dados do estudante "isaque", como matrícula, curso e faltas iniciais, e o programa confirma que o aluno foi salvo com sucesso no arquivo alunos.txt.

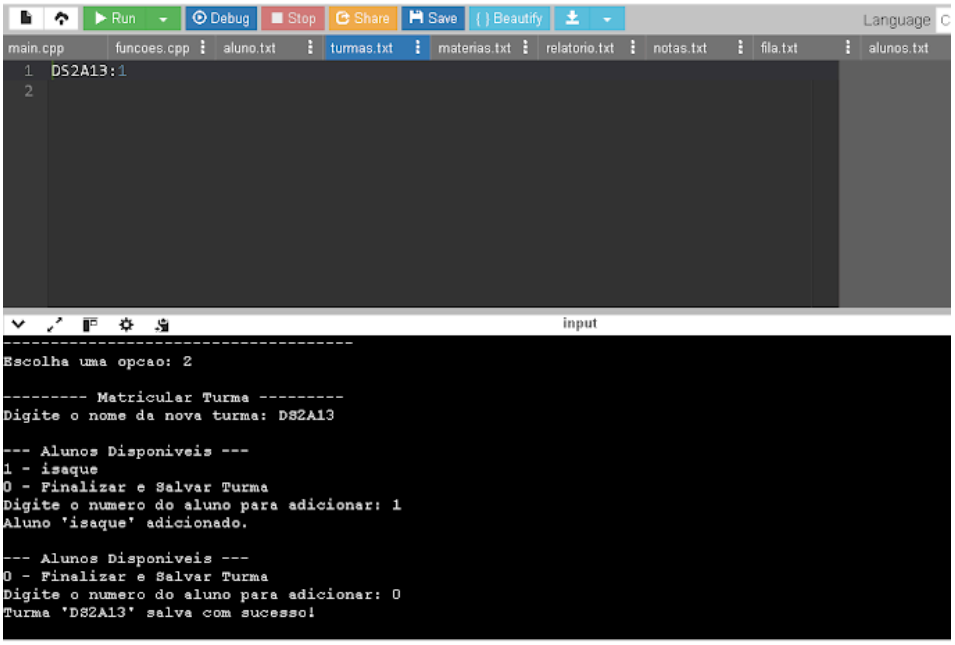


Imagem 3: Matricular Turma (Opção 2)

Com um aluno já cadastrado, o usuário escolhe a opção 2 para criar uma turma. Ele dá um nome à turma ("DS2A13"), adiciona o aluno "isaque" a ela e finaliza a operação, salvando a nova turma no arquivo turmas.txt.

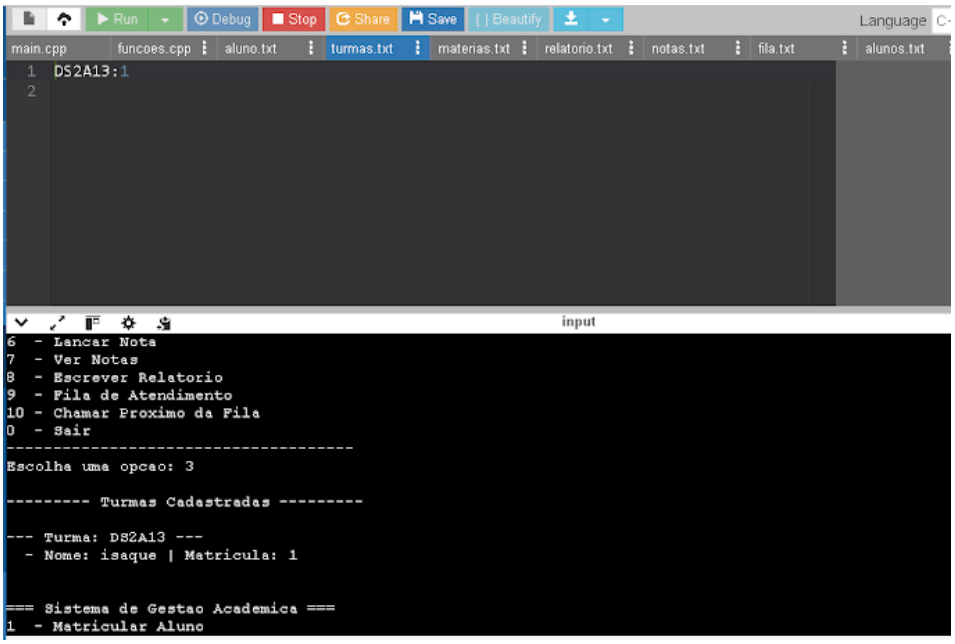


Imagem 4: Ver Turmas (Opção 3)

O usuário seleciona a opção 3 para visualizar as turmas existentes. O sistema lê os arquivos de turmas e alunos e exibe na tela a turma "DS2A13" com os detalhes do aluno "isaque", confirmando que a matrícula na turma foi bem-sucedida.

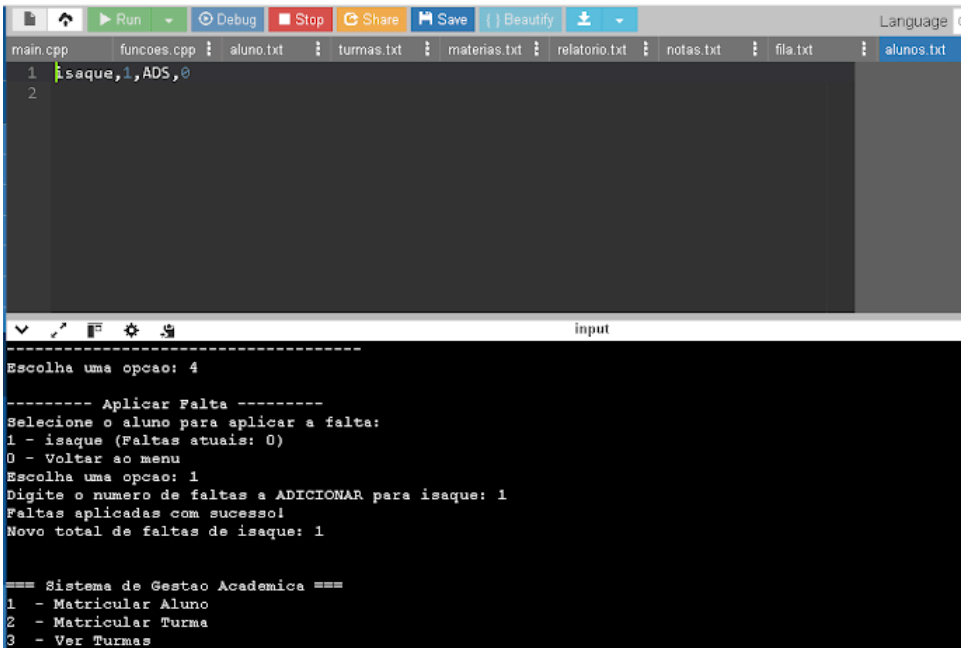


Imagem 5: Aplicar Falta (Opção 4)

Para gerenciar a frequência, o usuário escolhe a opção 4. O sistema lista os alunos e permite que o usuário adicione uma falta para "isaque", atualizando seu registro de ausências e salvando a alteração no arquivo.

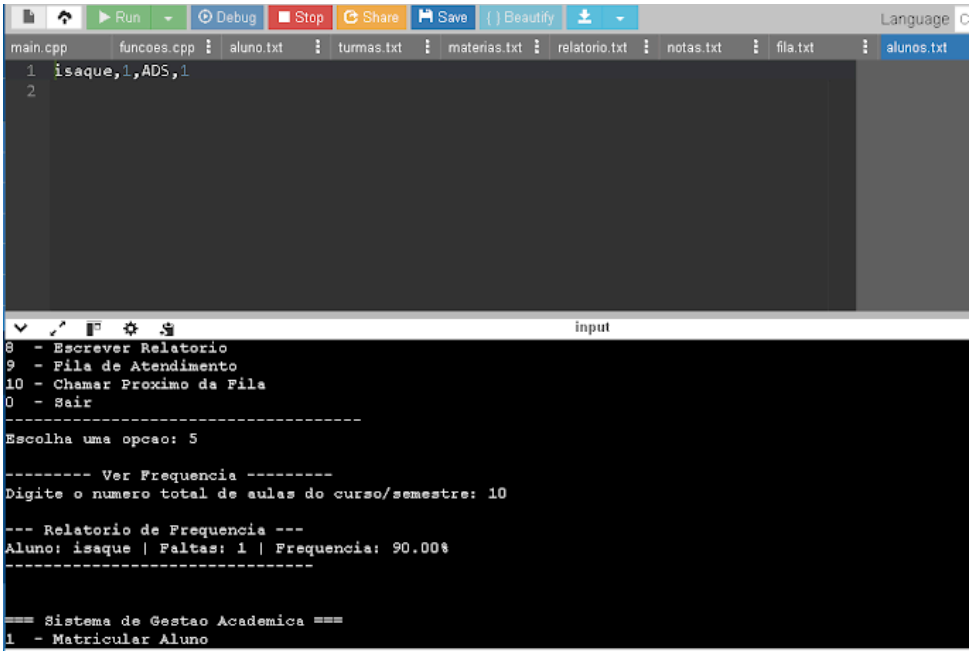


Imagem 6: Ver Frequência (Opção 5)

Utilizando a opção 5, o usuário verifica o percentual de frequência do aluno. Com base na falta aplicada e em um total de 10 aulas, o sistema calcula e mostra que a frequência de "isaque" é de 90%.

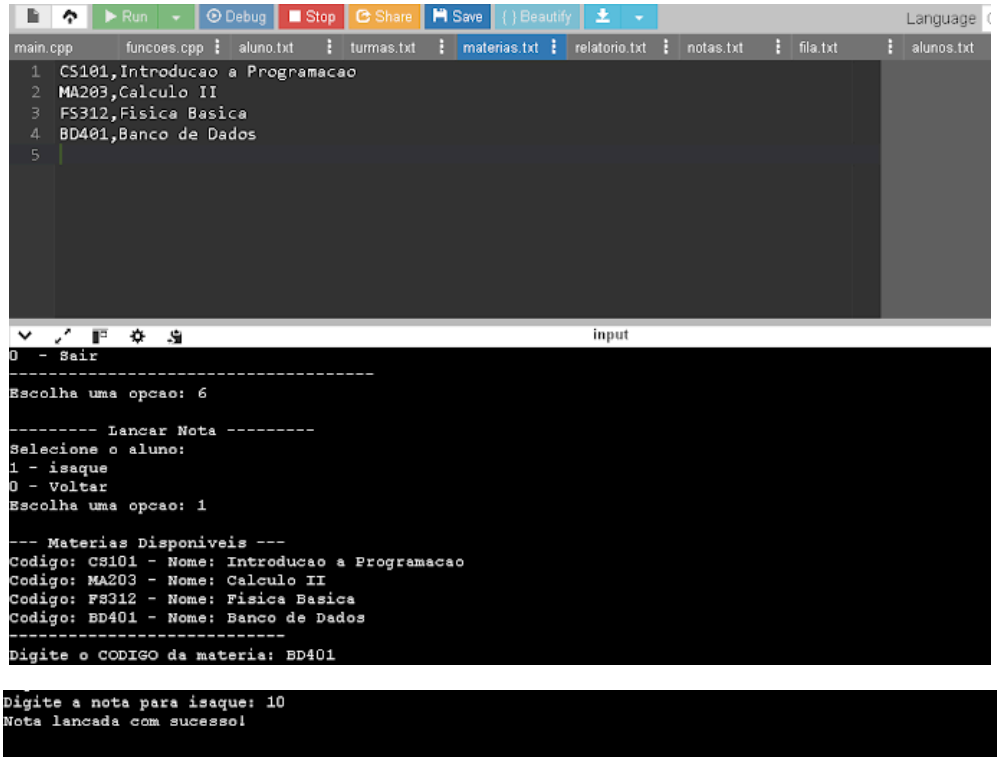


Imagem 7 e 12: Lançar Nota (Opção 6)

O usuário seleciona a opção 6 para lançar uma nota. Ele escolhe o aluno "isaque", seleciona a matéria "Banco de Dados" (BD401), digita a nota 10 e o sistema confirma que o lançamento foi salvo com sucesso no arquivo notas.txt.

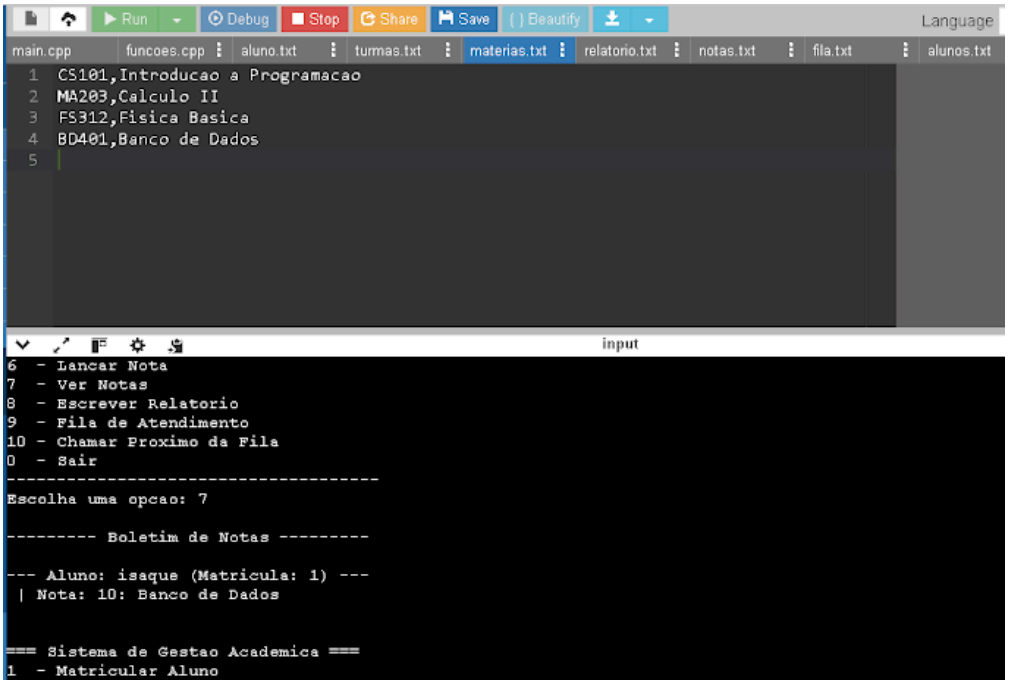


Imagem 8: Ver Notas (Opção 7)

Para consultar o desempenho, o usuário escolhe a opção 7. O sistema gera um boletim, lendo os dados do arquivo notas.txt e exibindo a nota 10 em "Banco de Dados" para o aluno "isaque".

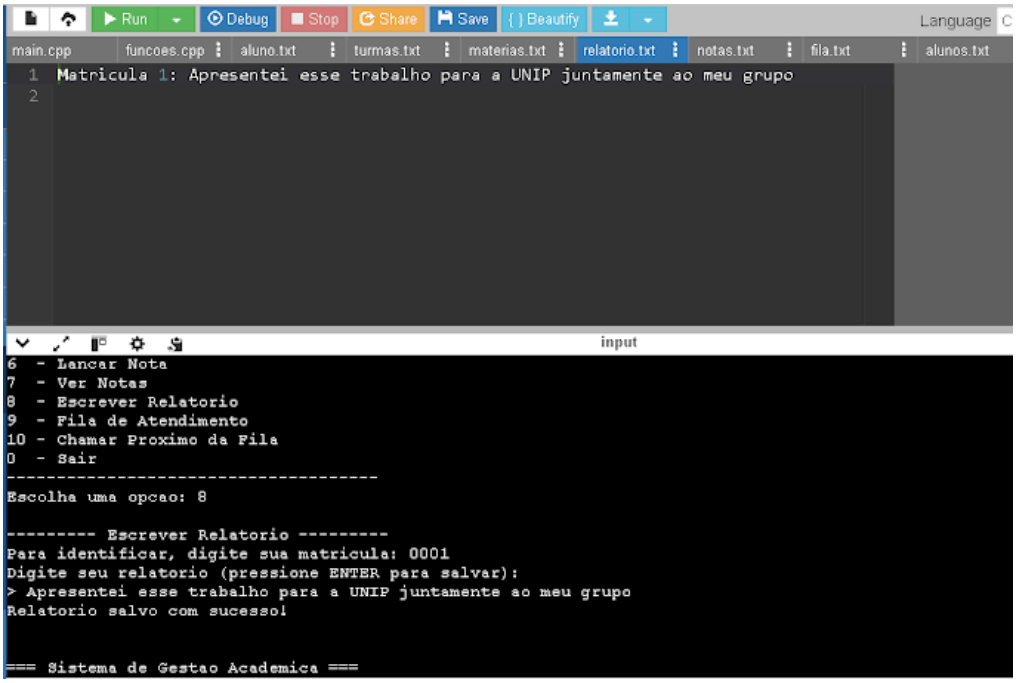


Imagem 9: Escrever Relatório (Opção 8)

O usuário utiliza a opção 8 para registrar uma observação. Identificando-se com a matrícula, ele escreve um texto que é salvo pelo sistema no arquivo relatorio.txt, servindo como um registro de atividades ou ocorrências.

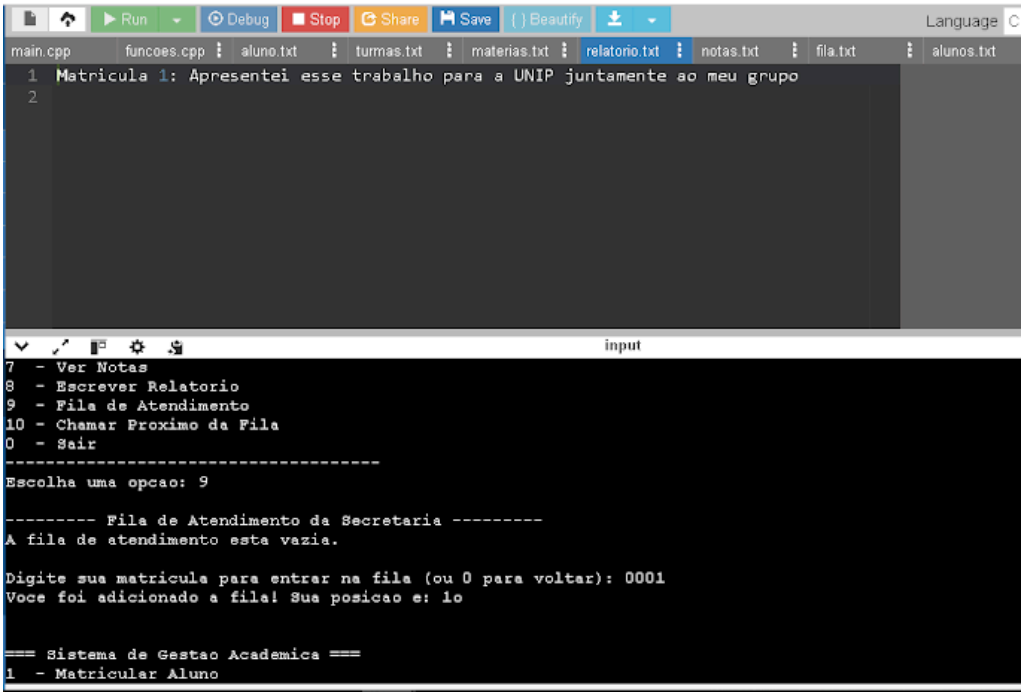


Imagem 10: Fila de Atendimento (Opção 9)

Demonstrando a função de atendimento, o usuário escolhe a opção 9. Como a fila está vazia, o aluno "isaque" insere sua matrícula e é adicionado como o primeiro da fila, com o status salvo no arquivo fila.txt.

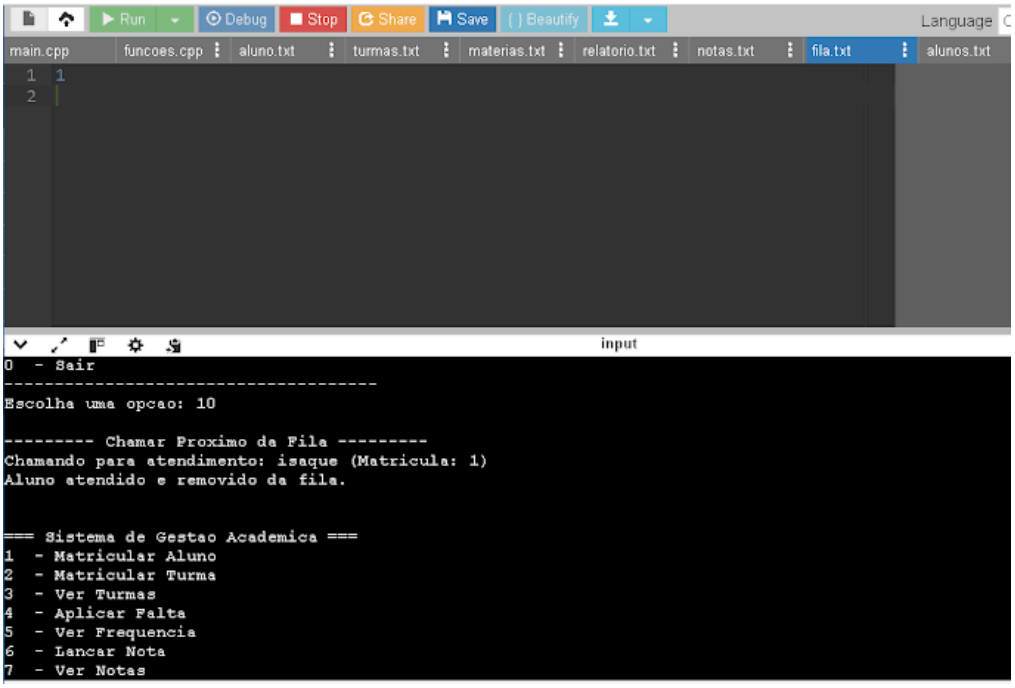


Imagem 11: Chamar Próximo da Fila (Opção 10)

Para dar andamento ao atendimento, o usuário seleciona a opção 10. O sistema chama o primeiro da fila, "isaque", e o remove, confirmando que o atendimento foi realizado e que a fila foi atualizada.

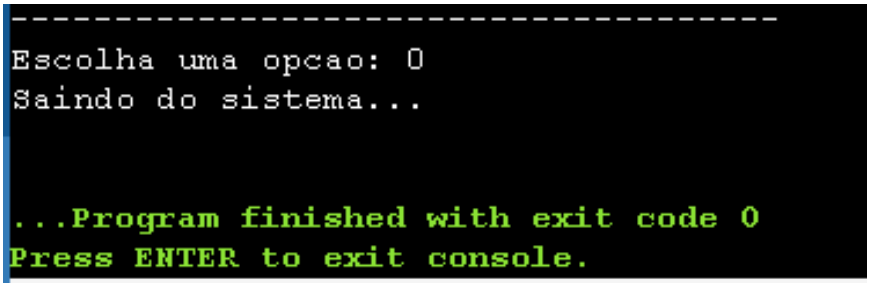


Imagem 13: Sair do Sistema (Opção 0)

Finalmente, o usuário escolhe a opção 0 para encerrar suas atividades. O programa exibe uma mensagem de despedida e finaliza a execução de forma limpa e segura.

Conclusão sobre a Finalidade do Código

Em resumo, os códigos apresentados servem para construir um sistema de gestão acadêmica completo, desde o acesso seguro até a execução das tarefas diárias. O projeto digitaliza e organiza operações essenciais de uma instituição de ensino, garantindo que informações sobre alunos, turmas e desempenho sejam salvas e administradas de forma estruturada e persistente.

Links de hospedagens:

Script Cadastro:
https://docs.google.com/document/d/1bK3bXCR1Tv2sZlqHoKnZf2ty5bkBH0-2PtxBvztxneU/edit?usp=sharing.

Script Funções:
https://docs.google.com/document/d/17fkzjpeeePYKzV4_L4ulKQsT4eRHHX9vwmVatL9QZoc/edit?usp=sharing.

Site dos arquivos JSON:

https://github.com/json-c/json-c/wiki.

Fontes para o desenvolvimento do código:

Linguagem C++

CPLUSPLUS.COM. C++ Language Tutorial. Disponível em: https://www.cplusplus.com/doc/tutorial/.

Acesso em: 14 nov. 2025.

JSON (Formato de Dados)

JSON.ORG. Introduction to JSON. Disponível em: https://www.json.org/json-pt.html.

Acesso em: 14 nov. 2025.

Biblioteca cJSON (para parsing JSON em C/C++)

GITHUB. DaveGamble/cJSON: ultralightweight JSON parser in ANSI C. Disponível em: https://github.com/DaveGamble/cJSON.

Acesso em: 14 nov. 2025.