PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

PROGRAMAÇÃO MODULAR

INF1301

Trabalho 2 - Projeto do Sistema de Avaliação de Professores

INTEGRANTE	MATRÍCULA
FELIPE EDUARDO NUNES DA SILVA	2320615
LUCAS BRITO OLIVEIRA DA SILVA	2411907
PEDRO PITTA PAULINO	2410242
GABRIEL VALENTE FERREIRA PIRES	2310488
THEO JEAN-MARIE J LEMAIRE	2520420

PROFESSOR

FLAVIO HELENO BEVILACQUA E SILVA

Rio de Janeiro

22 de Setembro, 2025

1) Introdução

Este projeto propõe o desenvolvimento do *Filhos da PUC*, uma plataforma digital escrita em Python dedicada a facilitar e centralizar o processo de avaliação de professores por parte dos alunos. O aplicativo funcionará como uma ferramenta moderna e intuitiva, tal qual um fórum ou rede social semelhante ao X, projetado para coletar e organizar feedback de maneira eficiente, segura e, quando desejado, anônima. Porém, mais do que um mero sistema de avaliações, espera-se criar um espaço de interação feito especialmente para os alunos da PUC-Rio, onde seja possível, além de gerar avaliações, simplificar a experiência universitária por meio da criação de uma rede de compartilhamento de ideias, experiências e oportunidades entre os estudantes, democratizando o acesso a informações que podem fazer a diferença e auxiliar na tomada de decisões com impacto significativo em suas trajetórias acadêmicas e profissionais. Em outras palavras, esse projeto propõe a entrega do MVP daquilo que virá a ser, um dia, a rede social oficial dos alunos da PUC-Rio. Deve-se observar que o foco de especificar Requisitos, Análise e Projeto e Casos de Teste resume-se estritamente ao backend da aplicação, pois o frontend será um mero suporte feito de modo apenas a validar as lógicas do backend, não sendo o foco principal deste trabalho.

2) Requisitos

A seguir, estão definidos os requisitos da aplicação, que definem exatamente aquilo que ela irá ou não fazer.

- O usuário apenas poderá acessar o serviço caso possua uma conta e esteja logado.

- Para criar uma nova conta, deve-se provar vínculo de estudante com a PUC-Rio (por simplicidade, será exigida a matrícula e o e-mail institucional).

- Matrícula, nome completo, nome de usuário, e-mail institucional e curso, além do cadastro de uma senha, devem ser informados no ato de criação de conta.

- O aluno pode cadastrar, em outro momento, a lista de disciplinas que já cursou ou que está cursando no período vigente.

- A lista de disciplinas existentes é fechada, sendo sua manutenção/atualização de responsabilidade da conta admin.

- Ao cadastrar uma disciplina cursada, o aluno deve informar o(s) período(s), o(s) professor(es), dia(s) da semana e o(s) horário(s)

em que ela foi cursada. Isso tem como objetivo a criação e o rastreamento de turmas.

- Turmas também possuem uma lista de professores associados, gerada quando no cadastro das matérias que determinado professor leciona ou já lecionou.

- Dessas informações referentes à turma, ressalta-se que apenas uma referência à disciplina (seja seu código, por exemplo) devem ser salvas no perfil do aluno.

- Uma disciplina possui código, número de créditos, nome e descrição.

- Professores possuem nome, associação a um departamento, uma lista de matérias que já lecionou ou que está lecionando atualmente e uma lista de avaliações que o mencionaram.

- Professor não é usuário da aplicação, portanto não é possível que gere avaliações.

- Há apenas perfis de usuários para alunos, mas isso não deve impedir a criação de páginas referentes aos professores, às

turmas e às disciplinas, nas quais estejam dispostos seus respectivos dados.

- Todo aluno pode gerar uma avaliação.

- Avaliações possuem obrigatoriamente data e hora de sua geração, um título, um comentário e referência ao aluno que a gerou caso não seja anônima (nome e nome de usuário). Opcionalmente, pode-se ter uma ou mais categorias, turma alvo (representando o motivo da avaliação, a qual pode ser relativa a um professor, a um aluno da turma, à disciplina etc) e número inteiro de estrelas de 0 a 5.

- Todas as informações da conta de usuário do aluno e suas avaliações criadas podem ser acessadas em seu perfil.

- Perfis de alunos podem ser públicos ou privados.

- Uma solicitação de adicionar pessoa à rede (inspirando-se no formato do LinkedIn) deve ser enviada pelo remetente e aceita pelo destinatário para que aquele possa ter acesso às informações deste em seu perfil privado.

- Não há necessidade de solicitação caso o perfil seja público.

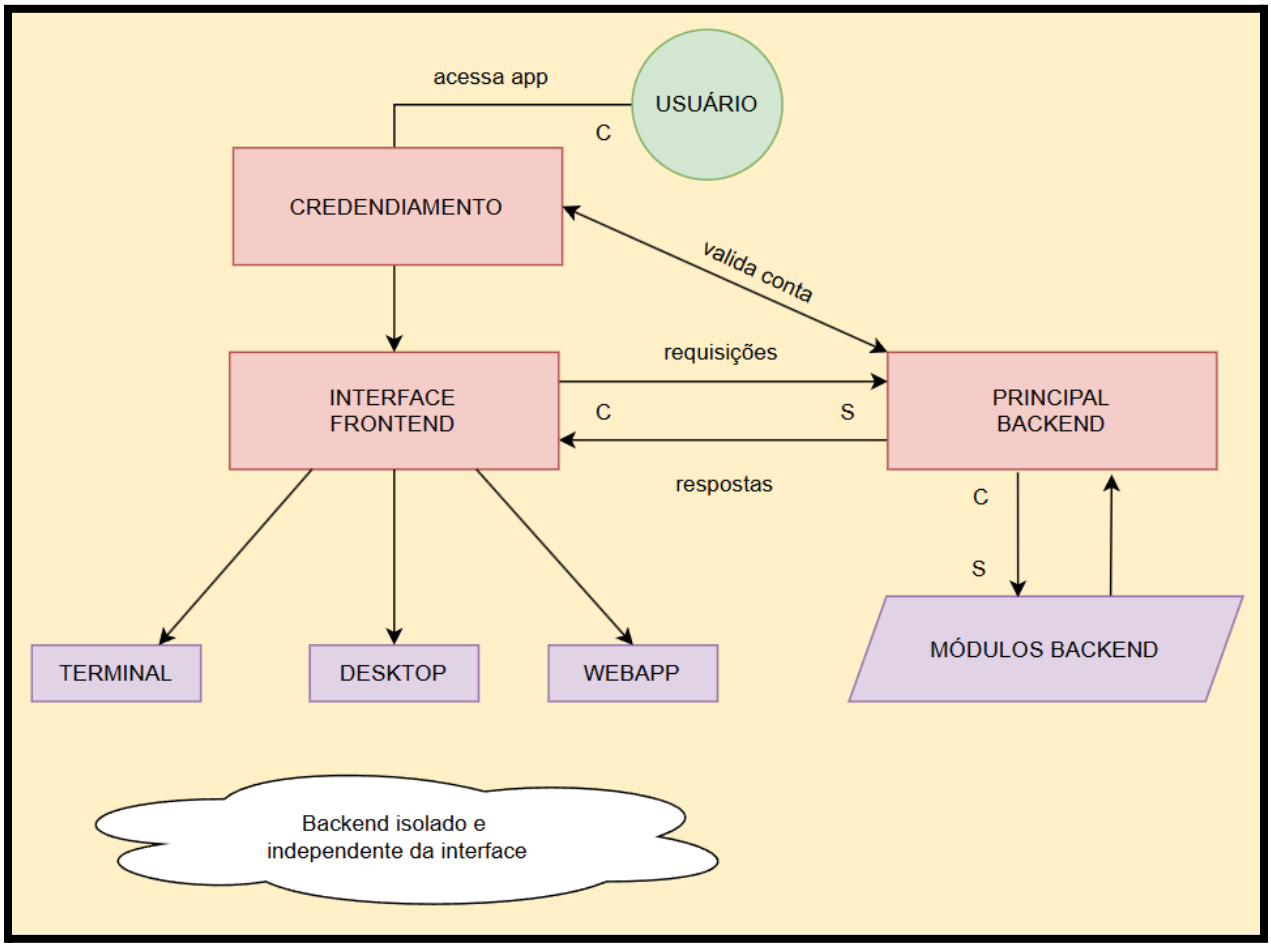
- A plataforma deve ser capaz de persistir os dados cadastrados pelos usuários, não somente em memória.

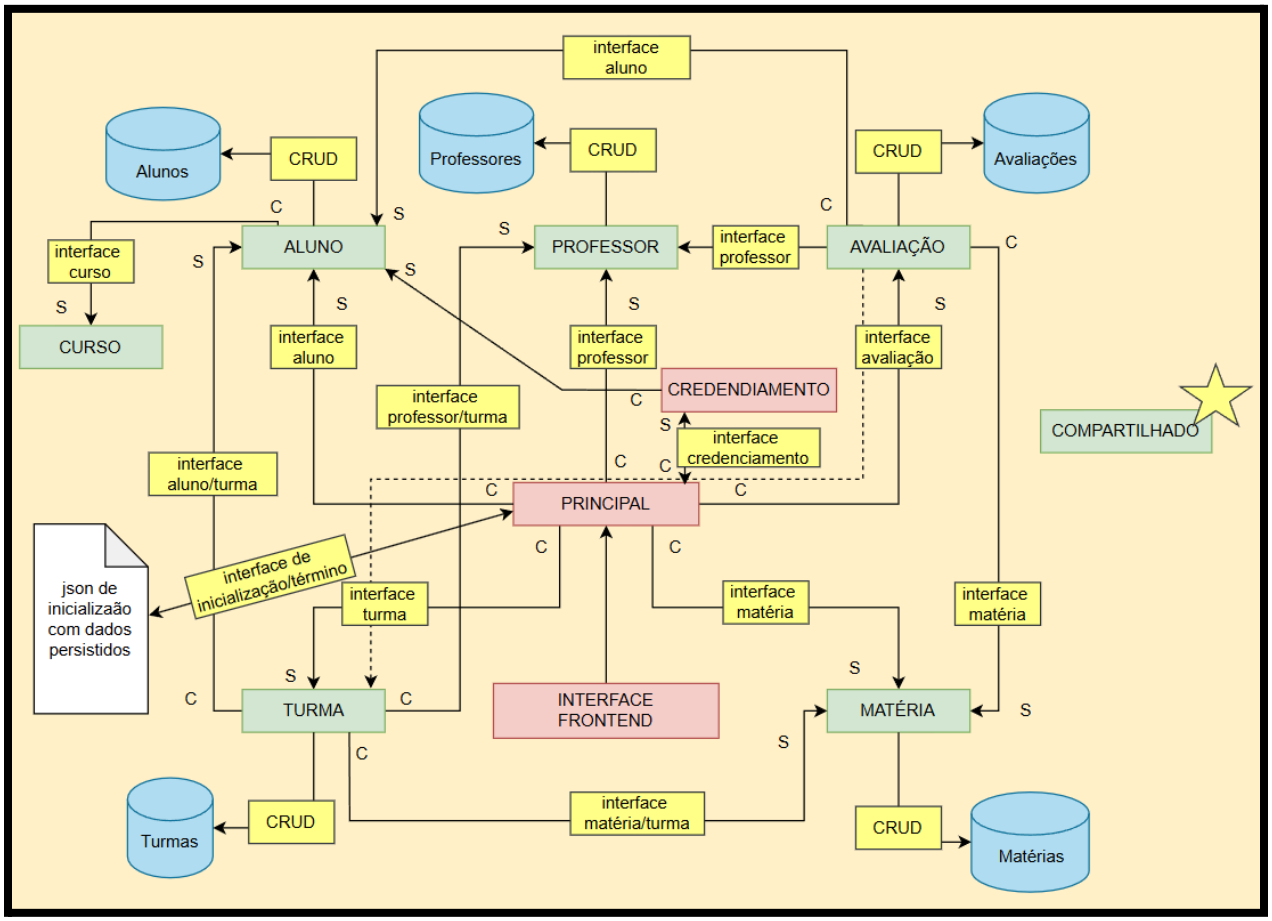
- O sistema não irá permitir que uma mesma matrícula seja usada para criar mais de uma conta de usuário.

3) Análise e Projeto

A etapa de análise e projeto será exposta abaixo por meio de diversas imagens e diagramas, nos quais são detalhados os protótipos de todas as funções, seus parâmetros e tipos de retorno.

- Diagrama de módulos da aplicação


acessa app
USUÁRIO
C
CREDENDIAMENTO
valida conta
requisições
INTERFACE
FRONTEND
C
S
PRINCIPAL
BACKEND
respostas
C
S
TERMINAL
DESKTOP
WEBAPP
MÓDULOS BACKEND
Backend isolado e
independente da interface



- Especificação dos módulos

Comentários gerais: funções de repositório possuem prefixo *repo* para indicar separação e isolamento entre manuseio de estruturas do repositório (no caso, as áreas de memória servindo de

armazenamento de informações) das funções que acessam essas estruturas, inspirado em princípios de arquitetura limpa e programação modular. Toda função de repositório com retorno inteiro retorna um booleano para indicar êxito ou fracasso na operação de CRUD sobre o repositório. Se for operação de leitura, retorna-se o tipo ou um vetor do tipo (a depender de ser leitura individual ou múltipla) que se está lendo. Sobre a persistência externa dos dados com um arquivo json, haverá uma função no principal dedicada a carregar esses dados e a serializá-los ao término da aplicação.

Aluno (aluno.c)
typedef struct Aluno
int matricula (pk)
char nome_de_usuario[MAX_USERNAME_LENGTH] (pk)
char senha[MAX_PASSWORD_LENGTH]
char nome[MAX_NAME_LENGTH]
char email_institucional[MAX_NAME_LENGTH]
Curso curso
Materia* materias[MAX_MATERIAS]
Avaliacao* avaliacoes[MAX_AVALS]
typedef AlunosDB
Aluno* alunosDB[MAX_ALUNO_COUNT]

funções públicas de acesso
int cria_aluno(Aluno* dados)
Aluno* busca_aluno(int matricula)
Aluno* busca_todos_alunos(void)
int troca_curso_aluno(int matricula, Curso curso)
int atualiza_aluno(int matricula, Aluno* dados)
int deleta_aluno(int matricula)
int valida_aluno(Aluno* dados)
int cria_materia_aluno(int matricula, int codigo_materia)
int aluno_fez_materia(int matricula, int codigo_materia)
Materia** busca_materias_aluno(int matricula)
int atualiza_materia_alunos(int codigo_velho, int codigo_novo)
int deleta_materia_aluno(int matricula, int codigo_materia)
int cria_aval_aluno(int matricula, Aval* dados)
Aval** busca_avals_aluno(int matricula)
Aval* busca_aval_aluno(int matricula, int id_aval)
int atualiza_aval_aluno(int matricula, int id_aval, Aval* nova_aval)
int deleta_aval_aluno(int matricula, int codigo_aval)

funções privadas
int repo_cria_aluno(Aluno* dados)
Aluno* repo_busca_aluno(int matricula)
int repo_atualiza_aluno(int matricula, Aluno* dados)
int repo_deleta_aluno(int matricula)
int repo_troca_curso_aluno(int matricula, Curso curso)
int repo_cria_materia_aluno(int matricula, int codigo_materia)
int repo_busca_materia_aluno(int matricula, int codigo_materia)
int repo_busca_materias_aluno(int matricula)
int repo_atualiza_materia_alunos(int codigo_velho, int codigo_novo)
int repo_deleta_materia_aluno(int matricula, int codigo_materia)
int repo_cria_aval_aluno(int matricula, Aval* dados)
Aval* repo_busca_aval_aluno(int matricula, int id_aval)
Aval** repo_busca_avals_aluno(int matricula)
int repo_atualiza_aval_aluno(int matricula, int id_aval, Aval* nova_aval)
int repo_deleta_aval_aluno(int matricula, int codigo_aval)

Curso
(curso.c)

typedef enum Curso

```
enum Curso {
        ADM,
        ARQ_URB,
        ART_CEN,
        CIEN_ECON,
        CIEN_SOCIOL,
        CIEN_COMP,
        COM_SOC,
        DSGN,
        DIR,
        ENG_AMB,
        ENG_CIV,
        ENG_COMP,
        ENG_CONTRLAUT,
        ENG_ELETR,
        ENG_IND,
        ENG_MEC,
        ENG_MAT,
        ENG_PROD,
        ENG_QUIM,
        PUBLI_COM,
        CINEM,
        COMUNIC_TEC,
```

```
        FARMAC,
        FILOS,
        FIS,
        GEO,
        HIST,
        IA,
        JORN,
        LET,
        MAT,
        MAT_APL,
        NEURO,
        NUTRI,
        PEDAG,
        PSI,
        QUIM,
        RI,
        SERV_SOC,
}
```

funções públicas de acesso

```
int valida_curso(Curso curso)
```

Compartilhado
(compartilhado.c)

definição de macros

```
#define MAX_LENGTH_NAME          100
#define MAX_MATERIAS             150
#define MAX_AVALS                100
#define MAX_ALUNO_COUNT          100
#define MAX_PROF_COUNT           100
#define MAX_AVAL_COUNT           1000
#define MAX_MAT_COUNT            1000
#define MAX_TUR_COUNT            1000
#define MAX_COMENT_LENGHT        1000
#define MAX_TITLE_LENGHT         100
#define MAX_USERNAME_LENGTH      200
#define MAX_MENTIONS_LENGTH      10
#define MAX_DESCRIPTION_LENGTH   500
#define MAX_PASSWORD_LENGTH      1000

#define SUCESSO                  0
#define ERRO                     1

enum Dia {
        SEG,
        TER,
        QUA,
        QUI,
```

```
        SEX,
        SAB,
        DOM
}
```

Departamento
(departamento.c)

typedef enum Depto

```
enum Depto {
        ADM,
        DAU,
        DAD,
        BIO,
        CIS,
        COM,
        JUR,
        ECO,
        EDU,
        CIV,
        ELE,
        IND,
        DEQM
        MEC,
```

```
        FIL,
        FIS,
        GEO,
        HIS,
        INF,
        LET,
        MAT,
        MED,
        PSI,
        QUI,
        SER,
        TEO,
        IRI
}
```

funções públicas de acesso

```
int valida_depto(Depto depto)
```

Professor (professor.c)	
typedef struct Prof	
int id (pk)	
char nome[MAX_NAME_LENGTH]	
Depto depto	
Materia materias[MAX_MATERIAS]	
Aval avaliacoes[MAX_AVALS]	// avaliacoes feitas POR ALUNOS (prof não avalia)
typedef ProfessoresDB	
Prof profsDB[MAX_PROF_COUNT]	
funções públicas de acesso	
int cria_prof(Prof* dados)	
Prof* busca_prof(int id)	
int atualiza_prof(int id, Prof* dados)	
int deleta_prof(int id)	
int valida_prof(Prof* dados)	
int cria_materia_prof(int id, int codigo_materia)	
int prof_ensina_materia(int id, int codigo_materia)	
int busca_materias_prof(int id)	

int atualiza_materia_profs(int codigo_velho, int codigo_novo)
int deleta_materia_prof(int id, int codigo_materia)
int cria_aval_prof(int id, Aval* dados)
Aval* busca_avaliacoes_prof(int id_prof)
Aval* busca_aval_prof(int id_prof, int id_aval)
int deleta_aval_prof(int id_prof, int id_aval)
int atualiza_aval_prof(int id_prof, int id_aval, Aval* nova_aval)
Prof* busca_todos_profs(void)
float calcula_aval_media_prof(int id) // média do número de estrelas, com 1 casa decimal apenas
funções privadas
int repo_cria_aluno(Aluno* dados)
Aluno* repo_busca_aluno(int matricula)
int repo_atualiza_aluno(int matricula, Aluno* dados)
int repo_deleta_aluno(int matricula)
int repo_cria_materia_aluno(int matricula, int codigo_materia)
int repo_busca_materia_aluno(int matricula, int codigo_materia)
int repo_busca_todas_materias_aluno(int matricula)
int repo_atualiza_materia_alunos(int codigo_velho, int codigo_novo)
int repo_deleta_materia_aluno(int matricula, int codigo_materia)
int repo_cria_aval_prof(int id_prof, Aval* dados)
int repo_deleta_aval_prof(int id_prof, int id_aval)
int repo_atualiza_aval_prof(int id_prof, int id_aval, Aval* nova_aval)
Aval** repo_busca_avals_prof(int id_prof)
Aval* repo_busca_aval_prof(int id_prof, int id_aval)

Matéria (materia.c)
typedef struct Materia
int codigo (pk)
int creditos
char nome[MAX_NAME_LENGTH]
char descricao[MAX_DESCRIPTION_LENGTH]
typedef MateriasDB
Materia materiasDB[MAX_MAT_COUNT]
funções públicas de acesso
int cria_materia(Materia* dados)
int existe_materia(Materia* dados)
Materia* busca_materia(int codigo)
int atualiza_materia(int codigo, Materia* dados)
int deleta_materia(int codigo)
int valida_materia(Materia* dados)
Materia* busca_todas_materias(void)
funções privadas
int repo_cria_materia(Materia* dados)
Materia* repo_busca_materia(int codigo)
int repo_atualiza_materia(int codigo, Materia* dados)
int repo_deleta_materia(int codigo)
float calcula_aval_media_materia(int codigo) // média do número de estrelas, com 1 casa decimal apenas

Turma (turma.c)
typedef struct Horario
Dia dia
int hora_inicio
int hora_fim
typedef struct Turma
int codigo (pk)
Materia materia
Horario* horarios // formato: [DIA_SEMANA, HORA_INICIO, HORA_FIM] , ex: [[SEG, 7, 9], [QUI, 14, 15]]
int periodo // ex: 20241, 20242, 20251, 20252 etc
Prof profs[MAX_PROF_COUNT]
Aluno alunos[MAX_ALUNO_COUNT]
typedef TurmasDB
Turma* turmasDB[MAX_TUR_COUNT]
funções públicas de acesso
int cria_turma(Turma* dados)
int existe_turma(Turma* dados)
Turma* busca_turma(int cod_turma)
int atualiza_turma(int cod_turma, Turma* dados)
int deleta_materia(int cod_turma)
int valida_turma(Turma* dados)
Turma* busca_todas_turmas(void)
funções privadas
int repo_cria_turma(Turma* dados)
Turma* repo_busca_turma(int codigo)
int repo_atualiza_turma(int codigo, Turma* dados)
int repo_deleta_turma(int codigo)

Avaliação
(avaliacao.c)

typedef enum CatAval

```
enum CatAval {
	PROF_BOM, // avaliação de elogio ao professor
	PROF_RUIM, // crítica ao professor
	PROF_MALUCO,
	TURMA_CHATA,
	TURMA_SALA_DE_AULA,
	ALUNO_INSUPORTAVEL,
	ALUNO_GENIAL,
	ALUNO_BURRO,
	ALUNO_PSICOPATA,
	ALUNO_ENGRACADO,
	MATERIA_DIFICIL,
	MATERIA_FACIL,
	MATERIA_NADA_A_VER,
	MATERIA_CHATA,
	MATERIA_PROVA_IMPOSSIVEL,
	MATERIA_PROVA_FACIL,
	MATERIA_MUITOS_TRABALHOS,
	OUTRO, // categoria indefinida porém com referência a alguma turma

	// abaixo, categorias que aceitam turma == NULL para se falar de qualquer outra coisa da PUC
	 ASSUNTOS_GERAIS,
	BANDEJAO,
	ASSALTO,
	BICICLETARIO,
	GINASIO,
	SPOTTED,
	PUC_URGENTE,
	VILA_DOS_DIRETORIOS,
	EVENTOS,
	ESTACIONAMENTO,
	BARRAQUINHAS_AZUIS,
	COMIDA, // onde comer, o que comer, preços... bem relevante inclusive
	OPORTUNIDADES_ESTAGIO,
	OPORTUNIDADES_TRABALHO,
	OPORTUNIDADES_PESQUISA,
	SEMANA_DA_INFORMATICA,
	CAINF,
	DEP_INFORMATICA,
	LAB_GRAD,
	SUGESTAO_AOS_DESENVOLVEDORES
}
```

typedef struct Aval
int id_aval (pk)
Aluno aluno_avaliador
struct tm data_aval // formato: struct tm proveniente de <time.h>
Turma turma_alvo
char mencoes[MAX_MENTIONS_LENGTH] // formato: p0a0a1a2a3m0 ==> menciona o prof índice zero, 4 primeiros alunos e a matéria
int estrelas // de 0 a 5
char comentario[MAX_COMENT_LENGHT]
char titulo_aval[MAX_TITLE_LENGHT]
CatAval* categoria // exemplo: AVAL_PROF, AVAL_MAT, AVAL_ALUNO, AVAL_TURMA, AVAL_HORARIO, AVAL_PROVA_DIFICIL, AVAL_PROF_MALUCO etc
char anonimo // booleano para avaliação anônima: 1 não exibe o nome, 0 exibe normalmente
typedef AvalsDB
Aval* avalsDB[MAX_AVAL_COUNT]
funções públicas de acesso
int cria_aval(Aval* dados)
Aval* busca_aval(int cod_aval)
int atualiza_aval(int cod_aval, Aval* dados)
int deleta_aval(int cod_aval)
int valida_aval(Aval* dados)
Aval* busca_todas_avals(int matricula_aluno_avalliador)
int valida_cat_aval(int matricula_aluno_avalliador)
funções privadas
int repo_cria_aval(Aval* dados)
Aval* repo_busca_aval(int id_aval)
int repo_atualiza_aval(int id_aval, Aval* dados)
int repo_deleta_aval(int id_aval)

- Relacionamento de funções entre módulos

Módulo	Chama funções de	Objetivo
principal	todos	inicializa, orquestra e finaliza o sistema

credenciamento	aluno	valida dados de um aluno no ato de criação de conta na plataforma
aluno	matéria, avaliacao	associa matérias e avaliações ao aluno
professor	matéria, avaliacao	associa matérias e recebe avaliações
turma	matéria, professor, aluno	compõe turmas completas
materia	-	módulo autônomo (lista de disciplinas)
avaliação	aluno, professor, turma, materia	avalia entidades e relaciona todas
frontend	principal e credenciamento	interface de entrada/saída de dados

- Definição do desenvolvedor responsável por cada módulo

4) Casos de Teste

Int cria_aluno(Aluno* dados)

Teste:

T1 - Retorna 1 (Sucesso) - Todos os Parâmetros são passados corretamente e nenhuma informação falta.

T2 - Retorna 0 (Falha) - Há uma falha na criação de Aluno por erro na definição da estrutura Aluno.

T3 - Retorno 0 (Falha) - Há uma falha na alocação dos dados por tipo de dados errados fornecidos.

T4 - Retorna 0 (Falha) - Dados incompletos ou campos obrigatórios nulos.

Aluno* busca_aluno(int matricula)

Teste:

T1 - Retorna Aluno com matrícula correspondente

T2 - Retorna ERRO - Busca informações de um aluno não registrado

T3 - Retorna ERRO - Incapaz de achar aluno por parâmetro errado passado

T4 - Retorna ERRO - Incapaz de achar aluno por formatação errada da matrícula passada

Aluno* busca_todos_alunos(void)

Teste:

T1 - Retorna todos os Alunos armazenados

T2 - Retorna nada - Incapaz de achar alunos por erro na alocação do vetor.

T3 - Retorna todos os Alunos armazenados - Acha todos os alunos mesmo com o parâmetro errado da função sendo passado, uma vez que o parâmetro não é usado para nada.

Int troca_curso_aluno(int matricula, Curso curso)

Teste:

T1 - Retorna 1 (Sucesso)

T2 - Retorna 0 (Falha) - Matricula passada em formatação errada

T3 - Retorna 0 (Falha) - Curso passado em formatação errada

T4 - Retorna 0 (Falha) - Tipo errado é passado como parâmetro

Int atualiza_aluno(int matricula, Aluno* dados)
Teste:

- T1 - Retorna 1 (Sucesso)
- T2 - Retorna 0 (Falha) - Matricula passada em formatação errada
- T3 - Retorna 0 (Falha) - Falha ao acessar dados do Aluno por matrícula inexistente.
- T4 - Retorno 0 (Falha) - Dados incompletos ou inválidos em Aluno* dados

Int deleta_aluno(int matricula)
Teste:

- T1 - Retorna 1 (Sucesso)
- T2 - Retorna 0 (Erro) - Matricula passada em formatação errada
- T3 - Retorno 0 (Erro) - Aluno não existe

Int valida_aluno(Aluno* dados)
Teste:

- T1 - Retorna 1 (Sucesso)
- T2 - Retorna 0 (Erro) - Matricula passada em formatação errada
- T3 - Retorna 0 (Erro) - Dados incompletos ou campos obrigatórios nulos.

Int cria_materia_aluno(int matricula, int codigo_materia)
Teste:

- T1 - Retorna 1 (Sucesso) - Matrícula e código de matéria válidos.
- T2 - Retorna 0 (Erro) - Matrícula inválida.
- T3 - Retorna 0 (Erro) - Código de matéria inválido.
- T4 - Retorna 0 (Erro) - Aluno ou matéria não existem.

Int aluno_fez_materia(int matricula, int codigo_materia)
Teste:

T1 - Retorna 1 (Sucesso) - Aluno completou a matéria.
T2 - Retorna 0 (Erro) - Aluno não fez a matéria.
T3 - Retorna 0 (Erro) - Matrícula ou código de matéria inválidos.
T4 - Retorna 0 (Erro) - Aluno ou matéria não existem.

Materia** busca_materias_aluno(int matricula)
Teste:

T1 - Retorna Matérias de acordo com a matrícula do aluno
T2 - Retorna ERRO - Matrícula Inválida ou Mal Formatada
T3 - Retorna ERRO - Aluno não encontrado ou vetor inacessível.
T4 - Retorna Vazio - Aluno existe mas não cursou nenhuma matéria.

Int atualiza_materia_alunos(int codigo_velho, int codigo_novo)

Teste:

T1 - Retorna 1 (Sucesso) - Atualiza corretamente todas as matérias com código antigo.
T2 - Retorna 0 (Erro) - Código antigo não encontrado.
T3 - Retorna 0 (Erro) - Código novo inválido ou duplicado
T4 - Retorna 0 (Erro) - Falha de alocação no vetor ou memória.

int cria_materia(Materia* dados)

Teste:

- T1 - Retorna 1 (Sucesso) - Todos os parâmetros na struct Materia são passados corretamente.
- T2 - Retorna 0 (Falha) - Ponteiro dados é NULO.
- T3 - Retorna 0 (Falha) - O código (pk) fornecido nos dados já pertence a uma matéria existente.

- T4 - Retorna 0 (Falha) - Dados inválidos ou campos obrigatórios nulos (ex: creditos negativo, nome vazio).

Materia* busca_materia(int codigo)

Teste:

- T1 - Retorna Materia* (Sucesso) - Retorna o ponteiro para a matéria com o código correspondente.
- T2 - Retorna NULL (Falha) - Busca por um código de matéria que não existe.
- T3 - Retorna NULL (Falha) - O parâmetro código passado é inválido (ex: negativo).

int atualiza_materia(int codigo, Materia* dados)

Teste:

- T1 - Retorna 1 (Sucesso) - Encontra a matéria pelo código e atualiza seus campos com os novos dados válidos.
- T2 - Retorna 0 (Falha) - O código informado não corresponde a nenhuma matéria registrada.
- T3 - Retorna 0 (Falha) - O ponteiro de dados fornecido para atualização é NULO.
- T4 - Retorna 0 (Falha) - Os novos dados fornecidos são inválidos (ex: créditos negativo).

int deleta_materia(int codigo)

Teste:

- T1 - Retorna 1 (Sucesso) - A matéria com o código correspondente é encontrada e removida.
- T2 - Retorna 0 (Falha) - O código informado não corresponde a nenhuma matéria registrada.

- T3 - Retorna 0 (Falha) - O parâmetro código passado é inválido (ex: negativo).

---

int cria_turma(Turma dados)

Teste:

T1 - Retorna 1 (Sucesso) - Todos os dados da turma são válidos, as matérias, professores e alunos referenciados existem, e a turma não é uma duplicata.

T2 - Retorna 0 (Erro) - A matéria especificada na Turma não existe.

T3 - Retorna 0 (Erro) - Um ou mais professores especificados na Turma não existem.

T4 - Retorna 0 (Erro) - Um ou mais alunos especificados na Turma não existem.

T5 - Retorna 0 (Erro) - Já existe uma turma idêntica (mesma matéria, período e professores).

T6 - Retorna 0 (Erro) - Dados da Turma são inválidos (ex: período em formato incorreto, horário nulo).

---

int existe_turma(Turma dados)

Teste:

T1 - Retorna 1 (Existe) - Uma turma com as mesmas características (matéria, período, professores) já está cadastrada.

T2 - Retorna 0 (Não Existe) - Nenhuma turma com as características informadas foi encontrada.

T3 - Retorna 0 (Erro) - Os dados fornecidos são inválidos para realizar a busca.

Turma* busca_turma(int cod_turma)

Teste:

T1 - Retorna Turma* - A turma com o código correspondente é encontrada e retornada.

T2 - Retorna NULL (Erro) - Nenhuma turma com o código especificado foi encontrada.

T3 - Retorna NULL (Erro) - O código da turma foi passado em um formato inválido (ex: negativo).

---

int atualiza_turma(int cod_turma, Turma dados)

Teste:

T1 - Retorna 1 (Sucesso) - A turma é encontrada e os novos dados são válidos.

T2 - Retorna 0 (Erro) - O cod_turma especificado não corresponde a nenhuma turma existente.

T3 - Retorna 0 (Erro) - Os novos dados (Turma dados) são inválidos (ex: referenciam uma matéria ou professor que não existe).

T4 - Retorna 0 (Erro) - O cod_turma foi passado em formato inválido.

---

int deleta_turma(int cod_turma)

Teste:

T1 - Retorna 1 (Sucesso) - A turma com o código especificado é encontrada e removida.

T2 - Retorna 0 (Erro) - Nenhuma turma com o código especificado foi encontrada.

T3 - Retorna 0 (Erro) - O cod_turma foi passado em formato inválido.

---

int valida_turma(Turma dados)

Teste:

T1 - Retorna 1 (Válido) - Todos os campos da Turma estão preenchidos corretamente e as referências (matéria, professores, etc.) são válidas.

T2 - Retorna 0 (Inválido) - A matéria referenciada não existe.

T3 - Retorna 0 (Inválido) - Pelo menos um dos professores referenciados não existe.

T4 - Retorna 0 (Inválido) - O formato do período é inválido (ex: 20253).

T5 - Retorna 0 (Inválido) - Os horários são nulos ou apresentam conflito.

Turma* busca_todas_turmas(void)

Teste:

T1 - Retorna Turma* - Retorna um ponteiro para o vetor de todas as turmas armazenadas.

T2 - Retorna NULL ou vazio - Nenhuma turma está cadastrada no sistema.

T3 - Retorna NULL (Erro) - Ocorre um erro de acesso ou alocação ao tentar buscar o vetor de turmas.

---

int repo_cria_turma(Turma dados)

Teste:

T1 - Retorna 1 (Sucesso) - A Turma é adicionada com sucesso ao banco de dados (ou estrutura de armazenamento).

T2 - Retorna 0 (Erro) - Falha ao alocar memória ou inserir os dados na estrutura de armazenamento.

Turma* repo_busca_turma(int codigo)

Teste:

T1 - Retorna Turma* - Encontra e retorna a turma correspondente ao código no banco de dados.

T2 - Retorna NULL - Não encontra nenhuma turma com o código especificado no banco de dados.

---

int repo_atualiza_turma(int codigo, Turma* dados)

Teste:

T1 - Retorna 1 (Sucesso) - Encontra a turma pelo código e atualiza seus dados no banco de dados.

T2 - Retorna 0 (Erro) - Falha ao encontrar a turma com o código especificado para realizar a atualização.

---

int repo_deleta_turma(int codigo)

Teste:

T1 - Retorna 1 (Sucesso) - Encontra a turma pelo código e a remove do banco de dados.

T2 - Retorna 0 (Erro) - Falha ao encontrar a turma com o código especificado para realizar a remoção.