Ministério da Educação
Universidade Federal do Cariri

UNIVERSIDADE FEDERAL DO CARIRI - UFCA
PRÓ-REITORIA DE GRADUAÇÃO - PROGRAD
CENTRO DE EDUCAÇÃO À DISTÂNCIA - CEAD
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ESPECIFICAÇÃO PROJETO 1 - POO
TEMA 6 – GERENCIADOR DE CURSOS E ALUNOS

1. Visão Geral

Desenvolver um sistema de **linha de comando (CLI)** ou **API mínima** (FastAPI/Flask, opcional) para gerenciar **cursos, turmas, alunos e matrículas**, com **controle de pré-requisitos**, **detecção de choque de horário**, **limite de vagas**, **frequência e notas**, além de **relatórios acadêmicos**. Persistência simples em **JSON** ou **SQLite**. O foco é aplicar **encapsulamento, herança, métodos especiais**, **validações rigorosas** e **relações entre múltiplas classes**.

2. Requisitos Funcionais

1. **Catálogo de cursos**
   - CRUD de **Curso** com: código (único), nome, carga horária, lista de **pré-requisitos** (códigos), ementa (opcional).
   - Impedir ciclos de pré-requisitos (ex.: A exige B e B exige A).
2. **Turmas (ofertas)**
   - Criar **Turma** com: id da turma, código do curso, período/semestre (ex.: 2025.2), dias/horários (ex.: `{"ter": "10:00-12:00", "qui": "10:00-12:00"}`), **vagas** (máx.), local (opcional).
   - Abrir/fechar turmas; impedir matrícula em turmas **fechadas**.
3. **Alunos**
   - CRUD de **Aluno** com: matrícula (única), nome, e-mail, histórico (disciplinas cursadas com nota e frequência).
   - Cálculo de **CR** (coeficiente/ média ponderada simples) com base no histórico.
4. **Matrícula**
   - **Matricular** aluno em turma, validando:

    - Pré-requisitos do curso **cumpridos** (no histórico).
    - **Choque de horário** com turmas já matriculadas no mesmo período.
    - **Vagas** disponíveis.
  - **Desistência/Trancamento** até data-limite (configuração).
5. **Acompanhamento acadêmico**
  - Lançar **frequência** (0–100%) e **notas** (0–10) por aluno em uma turma.
  - Calcular **situação** da matrícula: `APROVADO`, `REPROVADO_POR_NOTA`, `REPROVADO_POR_FREQUENCIA`, `CURSANDO`.
  - Regras de aprovação **parametrizáveis** (ex.: nota mínima 6.0; frequência mínima 75%).
6. **Relatórios**
  - Lista de alunos por turma, com vagas ocupadas vs. vagas totais.
  - **Taxa de aprovação** por curso e por turma.
  - **Distribuição de notas** por turma (média, mediana, desvio padrão).
  - **Alunos em risco** (nota parcial < corte ou frequência < mínimo).
  - **Top N** alunos por CR no período.
7. **Configurações** (`settings.json`)
  - `nota_minima_aprovacao`, `frequencia_minima`, `data_limite_trancamento` (YYYY-MM-DD), `max_turmas_por_aluno` (opcional), `top_n_alunos`.

3. Requisitos Técnicos de POO (RT)

- **Modelagem e herança:**
  - `Pessoa` (base) → `Aluno`.
  - `Oferta` (base) → `Turma` (pode incluir atributos/validações específicas).
  - `Curso`, `Matricula` (objeto que relaciona `Aluno` e `Turma` com notas/frequência/estado).
- **Encapsulamento e validações:**
  - Atributos sensíveis com `@property` (nota 0–10, frequência 0–100, CR ≥ 0, vagas ≥ 0).
  - Métodos para **matricular**, **lancar_nota**, **lancar_frequencia** com validações e mensagens de erro claras.
- **Métodos especiais (mín. 4):**
  - `Aluno.__lt__` para ordenação por CR (ou nome como critério secundário).
  - `Turma.__len__` retorna ocupação (qtd. de matrículas ativas).
  - `Curso.__str__`/`__repr__` para resumos.
  - `Matricula.__eq__` para comparação única (aluno+turma).
- **Regras de horário:**
  - Representar horários de forma comparável (tuplas de intervalos por dia) para detectar **choque**.

- **Persistência:**
  - Funções/módulo `dados.py` para salvar/carregar **cursos, turmas, alunos, matrículas** (JSON) ou uso direto de `sqlite3` (SQLite).
  - Rotinas de **seed** (pequeno conjunto de cursos/turmas/alunos) para demonstrar.
- **Testes (`pytest`):**
  - Pré-requisito não atendido → matrícula negada.
  - Choque de horário → matrícula negada.
  - Turma lotada → matrícula negada.
  - Lançamento de notas e cálculo de situação.
  - Distribuição de notas e taxa de aprovação.
  - Cálculo de CR e Top N.
- **Interface:**
  - CLI com subcomandos (`acad cadastrar-curso`, `acad abrir-turma`, `acad matricular`, `acad lancar-nota`, `acad relatorio taxa-aprovacao`) **ou** API mínima.

4. Regras de negócio (essenciais)

- **Pré-requisito:** aluno só pode cursar se todos os códigos de pré-requisito estiverem **aprovados** no histórico.
- **Choque de horário:** impedir matrícula se qualquer intervalo de dia/hora coincidir.
- **Limite de vagas:** não permitir matrícula quando `len(turma) == vagas`.
- **Trancamento:** permitido até `settings.data_limite_trancamento`; após isso, matrícula permanece ativa.
- **Aprovação:** `nota >= nota_minima_aprovacao` **e** `frequencia >= frequencia_minima`.
- **Limite de turmas por aluno (opcional):** negar matrícula acima do máximo definido.
- **Unicidade:** impedir duplicação de curso por código e de aluno por matrícula.

5. Critérios de aceite

✅ **POO**: herança (incluindo múltipla), encapsulamento com `@property`, **≥ 4 métodos especiais**

✅ **Regras:** pré-requisitos, choque de horário, vagas, trancamento e aprovação calculados corretamente.

✅ **Persistência:** JSON ou SQLite funcional e clara.

✅ **Documentação**: README com instruções, diagrama simples, decisões de design e cobertura de testes.

6. Cronograma

Formação dos Grupos - 05/12/2025

- Lista dos alunos de cada grupo.

Entrega 1 - 12/12/2025

- Lista das principais classes do sistema
- Descrição de responsabilidades de cada membro

Entrega 2 — 19/11/2025

- implementação inicial do sistema (mínimo 3 classes funcionando

Entrega Final — 17/01/2026

- Entrega final do sistema: código completo + documentação curta (README explicando classes, padrões usados e como rodar).

7. Entrega

A entrega de cada etapa será feita via AVA. Deve ser enviado o link do repositório do projeto no GitHub.

8. Avaliação

A avaliação será feita seguindo os seguintes critérios: Roteiro de Avaliação