CÂMPUS BAURU

COMPUTAÇÃO

**AFONSO[1], Cibele L. S. H.; YOSHIDA[2], Diego K.; SANTOS[3], Eduardo V.; DUARTE[4], Flavio C.; SILVA[5], Gabriel B.; ZANATA[6], Gabriel J. S.; PEREIRA[7], Leonardo L.; PINHEIRO[8], Thomas H. S.; SANTOS[9], Vinicius L.**

# DESENVOLVIMENTO E PROJETO DE SISTEMAS:

## sistema de gerenciamento de alunos e cursos

**Bauru, São Paulo**

**2025**

[1] 0.24100445ead@alunos.unisagrado.edu.br
[2] diego.24100039ead@alunos.unisagrado.edu.br
[3] eduardo.24100480ead@alunos.unisagrado.edu.br
[4] flavio.24100562ead@alunos.unisagrado.edu.br
[5] 0.24100411ead@alunos.unisagrado.edu.br
[6] gabriel.23100426ead@alunos.unisagrado.edu.br
[7] leonardo.24100327ead@alunos.unisagrado.edu.br
[8] thomas.24100016ead@alunos.unisagrado.edu.br
[9] vinicius.24100293ead@alunos.unisagrado.edu.br

## 1 INTRODUÇÃO

O projeto faz parte do módulo *Bootcamp: Desenvolvimento e Projeto de Sistemas*, e consiste no desenvolvimento uma *Application Programming Interface* (API) simples, tendo como proposta temática o gerenciamento de alunos e cursos de uma instituição de ensino.

## 2 PROPOSTA

A proposta foi disponibilizada em documentação à parte, conforme anexo A.

### 2.1 ESCOPO

O projeto abrange somente o processamento e retorno de requisições em nível do servidor *web*, sem a necessidade da implementação de interfaces gráficas, podendo-se fazer uso de outras ferramentas para visualização dos pacotes de dados enviados e recebidos. O gerenciamento dos dados deve ocorrer por meio do uso de Sistemas de Gerenciamento de Banco de Dados (SGBD), conectando-o à API no servidor *web*.

## 3. REQUISITOS

A coleta de requisitos foi realizada a partir da análise do documento contendo as instruções do projeto (vide anexo A).

### 3.1 REQUISITOS FUNCIONAIS

| Identificador | Requisito | Prioridade |
|---|---|---|
| RF001 | Cadastrar novos alunos | Essencial |
| RF002 | Adicionar alunos aos cursos | Essencial |
| RF003 | Listar alunos já cadastrados | Essencial |
| RF004 | Cadastrar novos cursos | Essencial |
| RF005 | Listar cursos já cadastrados | Essencial |
| RF006 | Listar alunos por curso (filtrar por curso específico) | Desejável |
| RF007 | Listar cursos em que determinado aluno está cadastrado (filtrar por aluno específico) | Desejável |

*Tabela 1 – Organização dos requisitos funcionais*

### 3.2 REQUISITOS NÃO-FUNCIONAIS

| Identificador | Categoria | Descrição | Prioridade |
|---|---|---|---|
| RNF001 | Projeto | A aplicação deverá utilizar uma das seguintes linguagens orientadas a objeto: *Python (Flask), Java Spring* ou *NodeJS.* | Essencial |
| RNF002 | Projeto | O banco de dados a ser utilizado deverá ser gratuito e do tipo relacional, dentre as opções: *SQLite, MySQL* ou *PostgreSQL.* | Essencial |
| RNF003 | Projeto | As requisições e retorno de dados deverão ser realizadas através de ferramentas como *Postman*, *Insomnia* ou similares. | Essencial |
| RNF004 | Previsibilidade | Os dados retornados e enviados deverão estar na estrutura esperada (corpo da requisição padronizado) | Essencial |
| RNF005 | Confiabilidade | Os dados enviados devem ser validados antes de gravados | Essencial |
| RNF006 | Confiabilidade | Não deve haver nenhuma falha no processamento das requisições (envio e recebimento) em todos os *endpoints* | Essencial |
| RNF007 | Portabilidade | A API poderá ser consumida em qualquer navegador e sistema operacional | Importante |
| RNF008 | Extensibilidade | A implementação de novas funcionalidades deverá ser facilitada, evitando dependências profundas e recursos obsoletos | Importante |
| RNF009 | Desempenho | O processamento de requisições não deve demorar mais que 400 ms | Desejável |

*Tabela 2 – Organização dos requisitos não-funcionais*

## 4 DIAGRAMAS UML (*UNIFIED MODELING LANGUAGE*)

A elaboração dos diagramas organiza as diferentes visões do projeto de maneira gráfica e intuitiva, de maneira a simplificar a implementação independente da tecnologia escolhida. Para este projeto, foram empregados os diagramas de: casos de uso e de classe.

### 4.1 DIAGRAMA DE CASOS DE USO

Esse diagrama recai sobre questões hipotéticas de implementação (como em uma possível expansão), visto que o escopo do projeto não abrange restrições de usuário, tanto de interface quanto de privilégios (na implementação real, qualquer usuário poderá realizar todos os tipos de requisição).

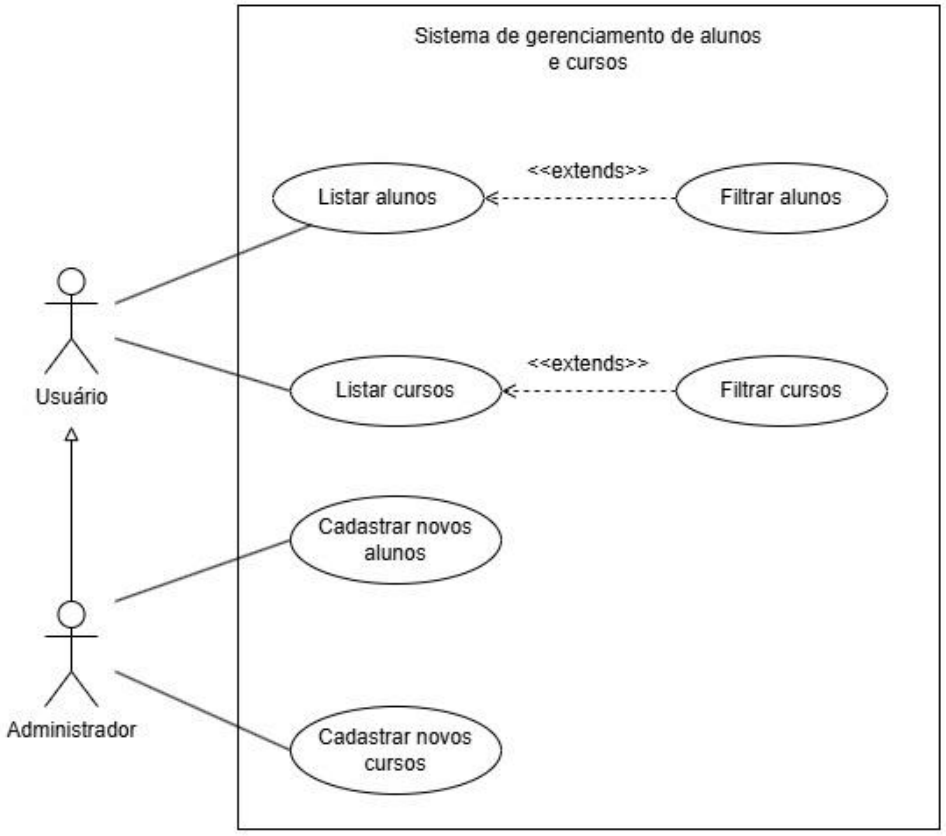


*Figura 1 – Diagrama de casos de uso*

## 4.2 DIAGRAMA DE CLASSE

Utilizamos, em maioria, classes já disponibilizadas em *Flask*. No entanto, foi necessário desenvolver uma nova classe para uso específico do projeto, com o intuito de gerenciar a comunicação com banco de dados, instanciando os objetos necessários e executando comandos SQL. A referida classe, *DBManager*, está definida de acordo com o diagrama abaixo:

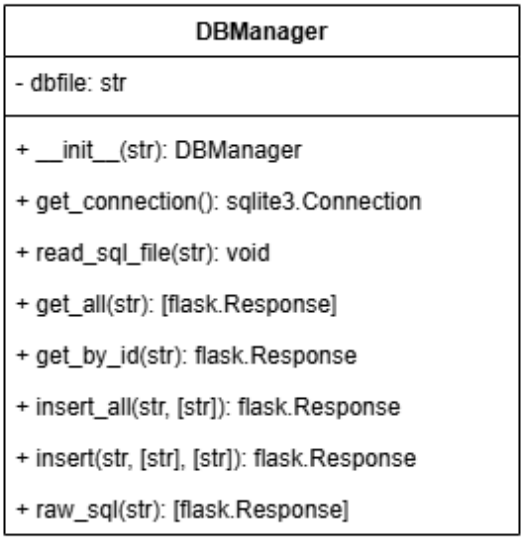


*Figura 2 – Diagrama de classe de DBManager*

| Nome | Descrição |
|---|---|
| *dbfile* | Atributo que contém o caminho para o arquivo .db (banco de dados SQLite). |
| *__init__()* | Construtor, inicializa o atributo *db_file* com o caminho fornecido. |
| *get_connection()* | Método que realiza a conexão da API com o banco de dados, retornando uma instância de *sqlite3.Connection*. |
| *read_sql_file* | Método que executa consultas SQL a partir de um arquivo *.sql*, localizado no caminho passado como parâmetro. Usado somente para inserção de dados. |
| *get_all()* | Método que retorna, em JSON, todos os dados da tabela especificada. |
| *get_by_id()* | Método que retorna, em JSON, o resultado de uma consulta em uma tabela especificada, pelo campo identificador. |
| *insert_all* | Método para inserção de dados em uma tabela especificada, em todos os seus campos. |
| *Insert()* | Método para inserção de dados em uma tabela especificada, somente nos campos informados. |
| *raw_sql()* | Método que executa comandos SQL informados diretamente no parâmetro de entrada, para consultas mais flexíveis. |

*Tabela 3 - Descrição das propriedades da classe DBManager*

## 5 MODELAGEM DO BANCO DE DADOS

Foi possível elencar, a partir dos requisitos extraídos, os atributos e entidades necessárias para o funcionamento da API, conforme os diagramas a seguir:

### 5.1 MODELO CONCEITUAL

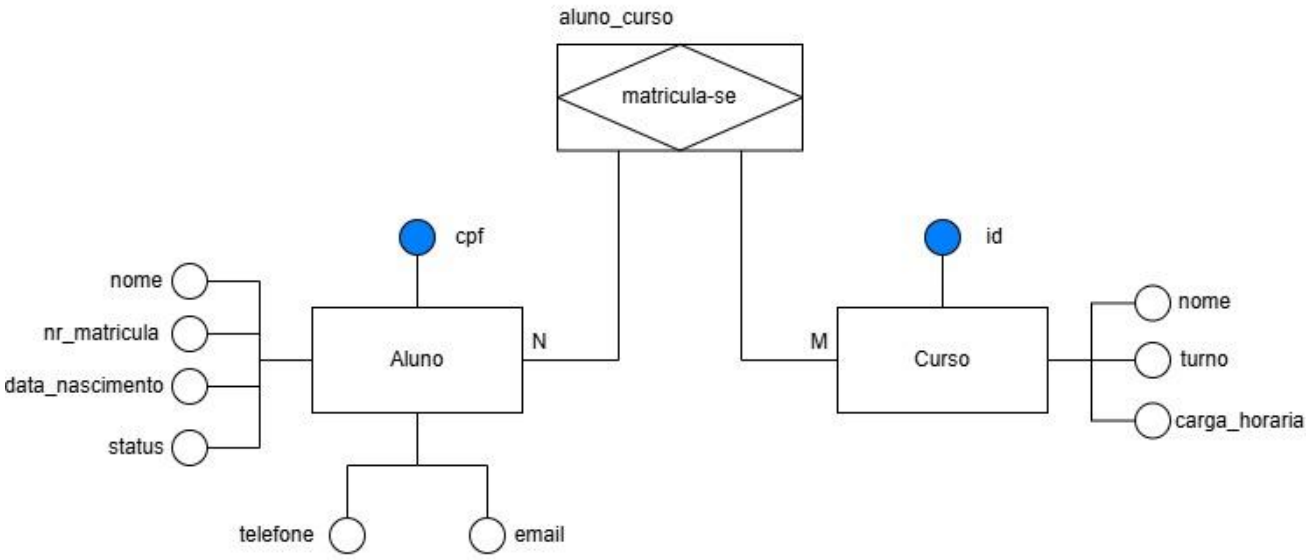


*Figura 3 – Diagrama conceitual, notação Peter Chen*

### 5.2 MODELO LÓGICO

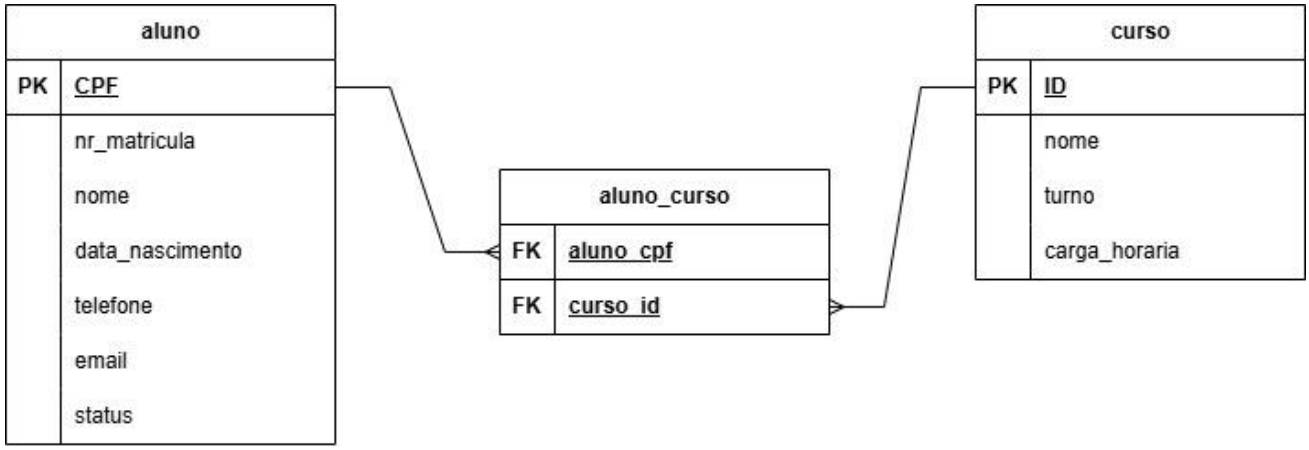


*Figura 4 – Diagrama lógico, com tabela de junção*

### 5.3 MODELO FÍSICO

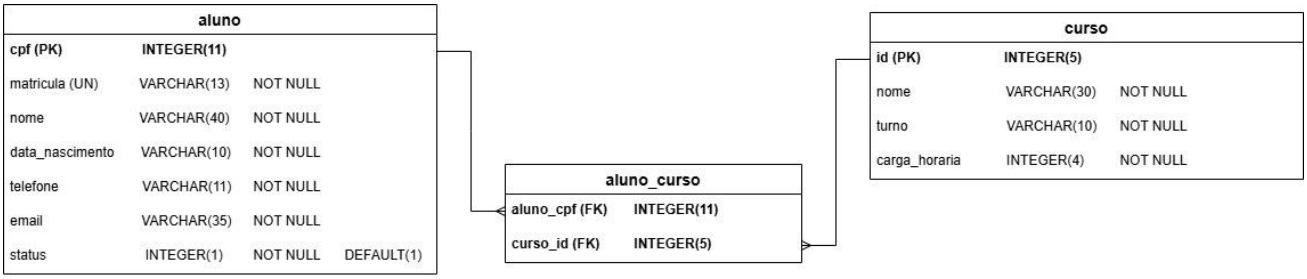


*Figura 5 – Diagrama físico, para o SGDB SQLite*

**ANEXO A –** Proposta do projeto disponibilizada pelo professor responsável

# Projeto: API Simples para Gerenciamento de Bootcamp

**1. Introdução**

Este projeto tem como objetivo criar uma API simples para gerenciamento. A API permitirá o cadastro e consulta de alunos e cursos.

**2. Objetivos**

- Criar uma API básica para gerenciar alunos e cursos.
- Permitir cadastro e listagem de alunos e cursos.

**3. Tecnologias Utilizadas**

**Sugestão de tecnologias:**

- **Linguagem**: Python (Flask); Java; Java Spring ou NodeJS.
- **Banco de Dados**: SQLite (simples e fácil de usar); MySQL; PostgreSQL

**Observação: não é permitido o uso de linguagens e/ou tecnologias que não foram citadas no documento. Seu uso acarretará nota zero.**

**4. Estrutura da API**

**4.1 Endpoints**

**Alunos**

- POST /alunos - Cadastrar um novo aluno.
- GET /alunos - Listar todos os alunos.

**Cursos**

- POST /cursos - Criar um novo curso.
- GET /cursos - Listar todos os cursos.

**Primeira entrega (31/03/25):** projeto no GitHub em seu estado atual.
**Segunda entrega (24/04/25):** projeto no GitHub (completo) e apresentação do projeto (obrigatória).

Datas e horários de apresentação:
26/04 pela manhã

**A data de apresentação está sujeita a alterações com aviso prévio.