RELATÓRIO

PROJETO FLASK INICIAL

Grupo: Guilherme Pereira Martins, Murylo Gomes Alves, João Vitor Simão Gomes

Data da atividade: 17/03/2025

Objetivo

O problema que estamos enfrentando é a criação de uma API para um CRUD de gerenciamento escolar, no qual cada entidade possui diversas funções para criar, deletar, atualizar e consultar dados.

1. Introdução

O sistema utiliza a linguagem de programação Python com o Flask para importar diversas bibliotecas, e empregamos o Venv para configurar um ambiente virtual para os testes. Além disso, realizamos o versionamento do código e a gestão de commits através do GitHub, garantindo que as bibliotecas e dependências do projeto sejam instaladas de forma isolada, sem interferir no sistema global.

Este tema é importante porque, ao utilizar ambientes virtuais, conseguimos evitar conflitos entre bibliotecas e dependências de diferentes projetos. Além disso, o versionamento de código com o GitHub permite que a equipe controle as alterações feitas no projeto, facilite a colaboração e evite perda de código ou erros ao reverter versões anteriores.

Esse processo é fundamental no mundo real, especialmente no desenvolvimento de software colaborativo, no qual as equipes precisam garantir que os projetos funcionem em qualquer máquina e manter um histórico de mudanças claro e controlado.

2. Descrição e Análise de Caso

Com o aumento da complexidade dos sistemas, é essencial entender como o desempenho de um programa pode ser impactado à medida que a quantidade de dados ou operações executadas cresce. Ou seja, desejamos saber como o programa se comporta à medida que a escala do problema aumenta. A análise de tempo de execução nos permite identificar gargalos e otimizar processos de forma a melhorar a performance da aplicação, garantindo sua escalabilidade.

No cenário da API que estamos desenvolvendo para o gerenciamento de uma escola, temos operações de criação, leitura, atualização e exclusão (CRUD) de dados para professores, alunos e turmas. Cada entidade possui características e operações que devem ser tratadas de forma eficiente para garantir um bom desempenho.

Algoritmo de Validação e Estruturação de Dados

A API inclui classes que representam as entidades de Professor, Aluno e Turma, e esses dados são manipulados por várias operações. A seguir, detalhamos algumas das melhorias que introduzimos com os novos algoritmos de validação:

Validação de Entradas: As classes foram projetadas para realizar validações antes de atribuir os valores aos atributos. Isso evita a inserção de dados inválidos, como nomes com mais de 100 caracteres, idades negativas ou incorretas, ou notas fora do intervalo permitido (0 a 10). Por exemplo:

- **Professor**: O nome não pode ter mais de 100 caracteres, e a idade deve ser um número inteiro positivo.

- **Aluno**: O nome não pode ser vazio, a idade deve ser positiva, e as notas devem estar entre 0 e 10.
- **Turma**: A descrição não pode ter mais de 100 caracteres, e o atributo "ativo" deve ser um valor booleano.

Estas validações são essenciais para garantir que os dados inseridos no sistema sejam consistentes, evitando erros na execução das operações de CRUD.

Estruturação de Dados e Métodos de Conversão: Cada classe tem um método to_dict(), que converte as instâncias das classes em dicionários. Isso facilita a conversão dos dados para o formato JSON, tornando-os mais adequados para serem retornados nas respostas da API. Essa estruturação também melhora a legibilidade e a organização do código.

Escalabilidade e Desempenho: A escalabilidade do sistema está diretamente ligada ao número de entidades manipuladas e ao tempo de execução das operações. Para melhorar o desempenho da API, é importante considerar as operações mais comuns (como as consultas GET, as atualizações PUT e as exclusões DELETE) e verificar se elas se comportam bem à medida que o número de registros aumenta.

- **Consultas**: À medida que o número de professores, alunos e turmas cresce, o tempo de resposta para consultar um dado específico (usando o ID, por exemplo) pode aumentar linearmente. Para melhorar isso, seria possível implementar uma abordagem com banco de dados e usar índices para otimizar as consultas.
- **Atualizações e Excluições**: A atualização de dados de um professor ou aluno, ou a exclusão de um professor ou turma, envolve a busca por um item em uma lista. Isso, em um grande número de registros, pode resultar em tempo de execução maior, já que estamos utilizando uma abordagem em memória (listas) sem otimizações como indexação ou particionamento de dados.

Estratégias de Melhoria: Para lidar com o crescimento do sistema e melhorar o desempenho, algumas estratégias podem ser adotadas:

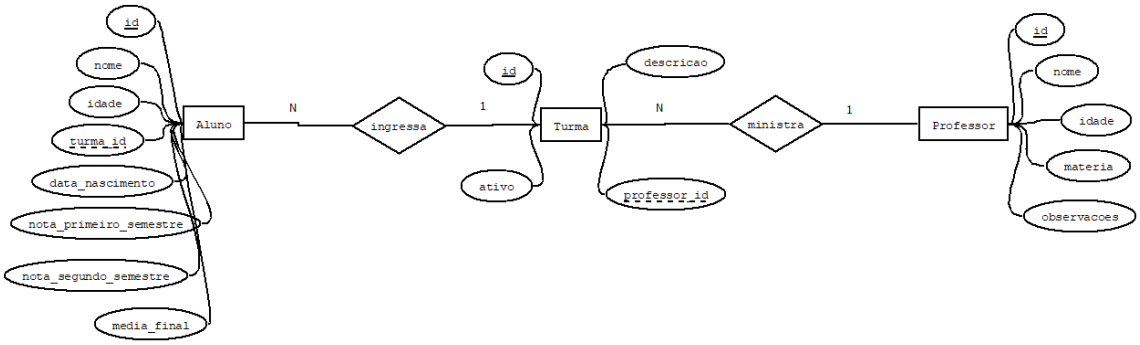
- **Utilização de banco de dados**: Em vez de manter as entidades em listas, seria mais eficiente usar um banco de dados para armazenar os dados das entidades. Isso possibilitaria consultas, atualizações e exclusões de maneira mais eficiente, além de permitir a persistência dos dados entre execuções.
- **Indexação**: Em sistemas maiores, indexar atributos como o id dos professores, alunos e turmas poderia melhorar significativamente o tempo de busca.
- **Paginação de resultados**: Quando a quantidade de dados for muito grande, uma estratégia comum é implementar paginação nas respostas da API, retornando apenas uma parte dos dados por vez.

Exemplo de Funcionamento do Algoritmo com Dados

Considerando o seguinte código para a criação de professores, turmas e alunos, temos:

- **Professores**: Cinco professores são criados com diferentes especializações e idades. O código implementa uma validação para garantir que as entradas sejam consistentes.
- **Turmas**: Cada turma é associada a um professor específico, com status ativo ou inativo. Essa estrutura permite a gestão de qual professor leciona em qual turma, facilitando a organização das informações.
- **Alunos**: Cada aluno está associado a uma turma e possui notas de dois semestres. A média final é calculada automaticamente, e o aluno tem suas informações devidamente validadas (nome, idade, notas).

Diagrama de Classe



3. Implementação ou Procedimento

A implementação ou procedimento de um sistema como o que está sendo desenvolvido para o gerenciamento de uma escola envolve várias etapas essenciais. A seguir, descrevo as etapas detalhadas que foram seguidas para criar a aplicação, desde a construção da API até a realização de testes automatizados.

Definição do Modelo de Dados

Primeiro, foi necessário definir os modelos de dados que representariam as entidades principais do sistema: **Professor**, **Aluno** e **Turma**. Cada uma dessas entidades possui atributos e comportamentos específicos que são fundamentais para o funcionamento do CRUD (Create, Read, Update, Delete).

- **Professor**: Contém atributos como nome, idade, disciplina, e observacoes.
- **Aluno**: Contém atributos como nome, idade, data_nascimento, nota_primeiro_semestre, nota_segundo_semestre, e media_final.
- **Turma**: Contém atributos como descricao, professor_id, alunos_ids e ativo.

Esses modelos foram definidos como classes Python com validações para garantir que os dados inseridos sejam válidos (como comprimento de texto e valores numéricos positivos).

Construção da API com Flask

A API foi construída utilizando o Flask, um microframework de Python para desenvolvimento de aplicações web. Cada entidade foi associada a um conjunto de endpoints para permitir as operações CRUD:

- **GET**: Para listar ou consultar um recurso específico.
- **POST**: Para criar um novo recurso.
- **PUT**: Para atualizar um recurso existente.
- **DELETE**: Para excluir um recurso.

Versionamento com Git

O código foi versionado utilizando **Git** e hospedado no **GitHub** para facilitar a colaboração e o controle das alterações no sistema. Com isso, é possível manter um histórico de mudanças, realizar revisões de código e facilitar o trabalho em equipe.

4. Resultados

Como resultado de todo o trabalho e implementação, temos uma estrutura de sistema bem consolidada, capaz de atender às necessidades de gerenciamento da escola de forma eficaz. A solução desenvolvida permite a manipulação dos dados por meio de um conjunto completo de operações CRUD (Create, Read, Update, Delete), que abrangem os principais recursos necessários para o gerenciamento de professores, alunos e turmas.

Além disso, a utilização do Flask para a construção da API proporcionou uma solução flexível e escalável, que pode ser facilmente expandida conforme as necessidades do sistema crescem.

Organização do Time

O time se organizou de forma a não sobrecarregar nenhum dos integrantes, de acordo com suas responsabilidades.

- **Guilherme Pereira Martins**: Responsável pela documentação, pela criação dos métodos PUT, pelos testes automatizados, pela organização da equipe em reuniões diárias e pela estruturação do GitHub.

- **Murylo Gomes Alves**: Um dos desenvolvedores do sistema, desenvolveu e auxiliou em diversos métodos do CRUD, de acordo com suas funções, além de organizar e estruturar o GitHub, e realizar os testes do CRUD.

- **João Vitor Simão Gomes**: Um dos desenvolvedores, iniciou o projeto estruturando e desenvolvendo-o de maneira promissora. João também auxiliou no restante do código.

Trabalhamos com a metodologia XP, na qual, todos os dias, realizávamos programação em par e reuniões diárias para acompanhar os projetos. Cada um elaborava um relatório diário para auxiliar no acompanhamento, organizando-se de forma que dois desenvolvedores trabalhavam no código e um ficava responsável pelos testes, considerando nossos horários de trabalho.

5. Conclusão

O trabalho desenvolvido ao longo do projeto resultou em uma aplicação bem estruturada e funcional, capaz de gerenciar as operações essenciais de um sistema escolar por meio de um CRUD robusto. A organização do time foi crucial para garantir que cada membro contribuísse de maneira eficaz, aproveitando suas habilidades e responsabilidades de forma equilibrada. A utilização de boas práticas de desenvolvimento, como o versionamento no GitHub e a criação de

testes automatizados, garantiu a qualidade do código e a manutenção de uma comunicação clara entre os integrantes.

Além disso, o processo de desenvolvimento foi acompanhado de perto por reuniões diárias, o que favoreceu a resolução de problemas de forma ágil e eficaz. A implementação de métodos PUT, a estruturação do sistema de testes e a documentação contribuíram para um produto final organizado, escalável e fácil de entender. Com isso, o projeto não só atendeu às expectativas, mas também se preparou para futuras melhorias e expansões como integrações com sistemas SGBD, estrutura de ETL, validações de acesso e desenvolvimento continuo de plataformas de gerenciamento. Em suma, o resultado foi um sistema sólido e eficiente, pronto para ser utilizado em um ambiente real de gerenciamento escolar.

6. Impacto e Conexão com o Mundo Real

A implementação deste sistema de gerenciamento escolar tem um impacto significativo, não apenas na organização interna de uma instituição educacional, mas também na forma como os dados são manipulados e processados no dia a dia de escolas e universidades. A solução proposta aborda desafios cotidianos enfrentados por escolas na administração de professores, alunos e turmas, automatizando e agilizando processos que antes eram manuais e demorados.

7. Desafios Futuros e Melhorias

Melhorar o sistema e integrá-lo com um banco de dados, além de criar uma interface para integrar os processos pela web, com formulários de entrada de dados. É importante também começar a implementar sistemas de segurança, autenticação e autorização dos usuários.

Além disso, continuar o desenvolvimento, facilitando os sistemas e automatizando o gerenciamento das escolas e universidades para manter um ambiente mais eficiente. O maior desafio a ser enfrentado é integrar esses sistemas nas universidades e realizar um estudo de mercado para integrar mais sistemas, com o objetivo de melhorar o funcionamento geral.

Referências

Modelo de Relatório – Faculdade Impacta

Franklyn Sancho, Medium. Análise de Algoritmos e um breve resumo sobre Análise Assintótica, 2021. Disponível em: https://franklyn-sanc.medium.com/an%C3%A1lise-de-algoritmos-e-um-breve-resumo-sobre-an%C3%A1lise-assint%C3%B3tica-f0efdfe6efb. Acesso em: 16/03/2025