FACULDADE IMPACTA
PROGRAMA DE PÓS-GRADUAÇÃO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Projeto Flask

Beatriz Bramont Oliveira
Giovanna Petrilli Venditti
Isadora Cristyne de Lima Silva
Larissa Costa Silva
Vinicios de Lima Basteiro

São Paulo
19/03/2025

Faculdade IMPACTA

Relatório 1

Relatório Projeto Flask - Entrega 1

Objetivo

Este projeto tem como objetivo apresentar a eficiência do framework Flask na linguagem Python, colocando em prática conceitos como Interface de Programação de Aplicações (API 's), dicionários, programação orientada a objetos, tratamentos de erro e TDD (Test-Driven Development).

1. Introdução

O Flask é um framework web utilizado em Python que facilita o desenvolvimento de API 's (Application Programming Interface), possui integração simples com banco de dados e outras tecnologias, e suporta testes automatizados, facilitando a implementação de um projeto. Por ser simples e flexível, ela é uma das ferramentas mais populares para a criação de interfaces, especialmente em ambientes educacionais ou protótipos rápidos. Uma de suas grandes vantagens é a facilidade de configurar rotas e manipular requisições HTTP, o que o torna ideal para projetos que buscam ensinar ou demonstrar conceitos técnicos de forma prática.

2. Descrição e Análise do Caso

Neste projeto foi estudado o benefício da utilização do framework na obtenção de dados correlacionados, através de um sistema amplamente utilizado no mundo real (gestão de um ambiente educacional). Para o desenvolvimento desta aplicação utilizamos o Visual Studio Code como ambiente de codificação, o

Postman para realizar os testes como GET e POST, o Git e o GitHub para compartilhamento e controle de versionamento dos códigos. Dividimos nosso repositório em alguns branches para separar os códigos em desenvolvimento e os que já estavam concluídos. Essa prática nos ajudou a assegurar que os programas funcionando perfeitamente não se perdessem no meio das edições de outras etapas ou por entre os commits.

3. Implementação ou Procedimento

O objetivo principal do projeto é criar um sistema para gerenciar informações sobre uma instituição educacional. O gerenciamento de dados dentro do código é um ponto importante do projeto tendo em vista que a API permite criar, ler, atualizar e (potencialmente) excluir informações sobre alunos, professores e turmas. Os dados são armazenados em um dicionário, que atua como um banco de dados em memória para fins de demonstração.

Endpoints dentro do código:

- **Criação (POST):**
  - /alunos (POST): Adiciona um novo aluno ao dicionário dici['alunos']. A verificação se o turma_id fornecido existe na lista de turmas.
  - /professor (POST): Adiciona um novo professor ao dicionário dici['professor'].
  - /turma (POST): Adiciona uma nova turma ao dicionário dici['turma']. A verificação se o professor_id fornecido existe na lista de professores.

- **Leitura (GET):**
  - /alunos (GET): Retorna uma lista de todos os alunos armazenados em dici['alunos'] como um JSON.
  - /professor (GET): Retorna uma lista de todos os professores armazenados em dici['professor'] como um JSON.

- /turma (GET): Retorna uma lista de todas as turmas armazenadas em dici['turma'] como um JSON.

- **Atualização (PUT):**
  - /alunos/<int:idAluno> (PUT): Atualiza os dados de um aluno específico, identificado pelo idAluno, no dicionário dici['alunos'].
  - /professor/<int:idProfessor> (PUT): Atualiza os dados de um professor específico, identificado pelo idProfessor, no dicionário dici['professor'].
  - /turma/<int:idTurma> (PUT): Atualiza os dados de uma turma específica, identificada pelo idTurma, no dicionário dici['turma'].

- **Deletar (DELETE):**
  - /alunos/<int:idAluno>(DELETE): Deleta os dados de um aluno, identificado pelo idAluno, no dicionário
  - /professor/<int:idProfessor>(DELETE): Deleta os dados de um professor, identificado pelo idProfessor, no dicionário
  - /turma/<int:idTurma>(DELETE): Deleta os dados de uma turma, identificada pelo idTurma, no dicionário.

Na utilização do Flask dentro da API tivemos as seguintes funções:

Flask(__name__): Cria uma instância do aplicativo Flask, que é o ponto de entrada para a API.

@app.route(): Este decorador é usado para associar URLs (rotas) a funções Python. Cada rota corresponde a um endpoint da API. Por exemplo, @app.route('/alunos', methods=['GET']) define que a função abaixo será executada quando uma requisição GET for feita para o endpoint /alunos.

request.jason: O objeto request do flak permite acessar os dados em JSON enviados no corpo das requisições POST e PUT.

jsonify(): A função jsonify converte os dados python (dicionários e listas) em formato JSON, que é o formato padrão para respostas de APIS REST.

app.run(debug=True): Inicia o servidor Flask, que fica escutando por requisições HTTP. O debug=True habilita o modo de depuração, que fornece informações úteis durante o desenvolvimento.

Fizemos a utilização de um dicionário python para realizar o armazenamento de dados da API, esse dicionário contém listas de dicionários, onde cada dicionário representa um aluno, professor ou turma. Isso é equivalente a ter uma "tabela" em memória para cada entidade existente na API.

A interação com algoritmos é feita por meio de busca linear, ou seja, verificando cada elemento de uma lista um a um. Dentro das funções updateAlunos, updateProfessores e updateTurma ela é usada para encontrar o registro com o id correspondente. A busca linear itera sobre a lista até encontrar o elemento desejado ou chegar ao final da lista.

Diante de um cenário onde irão ocorrer possíveis aumentos de entrada e saída de dados dentro da API, a busca linear terá um tempo de resposta proporcional a quantidade de de itens a se percorrer em cada uma das listas, visto que a pior hipótese diante dos casos será o ID não existir (nosso tratamento de erros irá devolver essa informação ao usuário), ou estar no final da lista, o que pode deixar o tempo de execução relativamente maior.

Uma das principais dificuldades encontradas no processo foi construir o arquivo de testes unitários, já que o sistema projetado é razoavelmente amplo e o grupo almejou construir testes práticos que abrangessem todo o programa, sem falhas ou brechas.

4. Resultados

Este projeto é completamente estruturado para receber informações do usuário, que pode ou não cometer deslizes ao enviar o JSON. Para evitar que esses erros passem despercebidos, o programa possui validações que garantem a integridade das informações e evitam o armazenamento de dados inválidos. Além disso, foi estabelecido um conjunto de testes para autenticação do funcionamento correto do programa.

No arquivo principal incluímos algumas validações, como por exemplo a “verificar_duplicacao” que verifica se o ID recebido no arquivo JSON já existe na API. Caso sim, ele envia uma mensagem de erro. Implementamos também a função “verificar_campo_null”, que valida se há no JSON alguma chave enviada com valor nulo, considerando que todos os campos são obrigatórios.

Alguns “dicionários” nesse sistema são interligados através do ID de cada entidade cadastrada, para que todos os registros sejam efetuados sem pendências, criamos variáveis que buscam o id da entidade relacionada . Por exemplo: na função “createAluno” é necessário o id do dicionário Turma, logo, para o cadastro do aluno ser válido, o JSON precisa conter um idTurma válido, pois sem a turma existir, não há como registrar o aluno. O mesmo procedimento acontece para turma, que necessita do id do professor.

Estruturamos o programa principal em rotas, utilizando o decorator @app.route, informando ao framework qual URL deve ser acionada para as funções, assim, separando o CRUD de cada entidade do nosso cenário.

Testes Realizados:

Os testes realizados tinham como objetivo verificar o comportamento da API em relação às operações de criação, leitura, atualização e exclusão (CRUD) de professores, turmas e alunos. Para isso, utilizamos a biblioteca unittest do Python e a biblioteca requests para realizar as chamadas HTTP aos endpoints da API.

Procedimentos de validação:

- **Testes de criação (POST):** Enviamos requisições POST para os endpoints /professor, /turma e /alunos com dados válidos e verificamos o código de status da resposta.
- **Testes de atualização (PUT):** Enviamos requisições PUT para os endpoints /professor/{id}, /turma/{id} e /alunos/{id} com dados atualizados e verificamos o código de status da resposta.
- **Testes de exclusão (DELETE):** Enviamos requisições DELETE para os endpoints /professor/{id}, /turma/{id} e /alunos/{id} e verificamos o código de status da resposta.
- **Testes de listagem (GET):** Enviamos requisições GET para os endpoints /professor, /turma e /alunos para verificar se os recursos criados foram listados corretamente.

Os testes foram executados e os resultados foram os seguintes:

- **Testes de Professor:** Os testes de criação, atualização e exclusão de professores foram realizados com sucesso.
- **Testes de Turma:** Os testes de criação, atualização e exclusão de turmas foram realizados com sucesso.
- **Testes de Aluno:** Os testes de criação, atualização e exclusão de alunos foram realizados com sucesso.

Os resultados atenderam às expectativas?

Sim, pelos testes que analisamos, parece que as operações básicas do **CRUD** (Create, Read, Update, Delete - Criar, Ler, Atualizar, Excluir) da API funcionaram como esperado. A maioria dos testes verificou se as ações de **criar**, **atualizar** e **excluir** estavam retornando os códigos de status HTTP que indicam sucesso. Além disso, os testes de **criar** também confirmaram que os dados estavam sendo armazenados e podiam ser **lidos** posteriormente na lista de recursos. Em resumo, os resultados parecem ter atendido às expectativas em relação às funcionalidades do CRUD da API.

O que eles revelam sobre o problema ou caso analisado?

Os resultados indicam que a API implementada para gerenciar professores, turmas e alunos está funcionando corretamente, mas é importante realizar testes mais abrangentes para garantir a qualidade e confiabilidade da API.

No entanto, é importante notar que estes são apenas testes de unidade básicos. Para uma avaliação mais completa do sistema, seriam necessários testes mais abrangentes, incluindo:

- **Testes de casos de erro:** Testes para verificar o comportamento da API em situações de entrada inválida, dados ausentes, tentativa de exclusão de recursos inexistentes, etc.
- **Testes de integração:** Testes que verificam a interação entre diferentes partes da API e com outros sistemas, se houver.
- **Testes de desempenho:** Testes para avaliar a capacidade de resposta da API sob diferentes cargas de trabalho.
- **Testes de segurança:** Testes para verificar a proteção da API contra vulnerabilidades.

Apesar disso, os testes fornecidos demonstram que as funcionalidades básicas de gerenciamento de dados para professores, turmas e alunos estão implementadas e, aparentemente, funcionando corretamente, conforme os critérios definidos nos testes unitários.

5. Conclusão

A API desenvolvida pelo grupo teve como objetivo a criação de uma API REST básica utilizando o framework Flask em Python, com o objetivo de gerenciar informações sobre uma instituição educacional usando rotas definidas com o decorador @app.route(), permitindo a interação com os dados através de requisições HTTP. Os dados são armazenados em um dicionário Python (dici), servindo como um banco de dados em memória para fins de demonstração.

Embora funcional, o nosso sistema apresenta limitações significativas em termos de escalabilidade e desempenho, especialmente em um ambiente de produção. O uso de busca linear nas funções de atualização resulta em um tempo

de execução proporcional ao tamanho dos dados, tornando-se ineficiente para grandes volumes. Além disso, o servidor de desenvolvimento Flask, usado por padrão, não é projetado para lidar com um grande número de requisições simultâneas. O que já nos leva a elaborar e desenvolver medidas futuras para a otimização destes dados

Em resumo, o código fornecido é um protótipo útil para entender os conceitos básicos de criação de uma API REST com Flask. No entanto, para ser utilizado em um ambiente de maior carga de dados, serão necessárias otimizações significativas em termos de escalabilidade do código, desempenho e robustez. A substituição do armazenamento em memória por um banco de dados, o uso de um servidor adequado e a implementação de cache são passos essenciais para transformar este protótipo em um sistema capaz de lidar com grandes volumes de dados e um grande número de requisições que ocorrerão de formas simultâneas.

6. Impacto e Conexão com o Mundo Real

A API está simulando um sistema básico de gerenciamento de dados, refletindo cenários práticos encontrados em sistemas de registro escolar. Ao seguir os princípios RESTful, a API utiliza métodos HTTP para operações CRUD, uma prática comum em aplicações web e móveis modernas que facilitam a comunicação entre sistemas. A inclusão de testes unitários com a biblioteca unittest destaca a importância dos testes automatizados para garantir a qualidade e confiabilidade das APIs em produção.

Fora do ambiente acadêmico, o aprendizado sobre criação e teste de APIs RESTful é crucial para desenvolvedores web e de backend, permitindo a construção de APIs que integram sistemas, fornecem dados para aplicativos móveis e criam microsserviços. A automação de tarefas, como coleta e processamento de dados, torna-se viável com APIs, otimizando fluxos de trabalho em diversas áreas. O conhecimento em testes unitários é essencial para garantir a qualidade do software.

A integração de sistemas legados com novos sistemas é facilitada por APIs, melhorando a troca de informações entre setores.

Em diversas áreas, as APIs desempenham um papel fundamental. Elas permitem o gerenciamento de informações de clientes, a integração de sistemas de pagamento e logística, e a criação de experiências de usuário interativas. Além disso, as APIs são usadas em sistemas de monitoramento em tempo real e sistemas de autenticação, demonstrando sua versatilidade e importância em diversas aplicações práticas.

7. Desafios Futuros e Melhorias

Entrando em consenso, o grupo acredita que a melhoria na gestão de tarefas é um ponto importante a ser levado em consideração nas próximas etapas do projeto, tendo em vista a suma importância de uma boa organização na divisão de responsabilidades. Outro ponto importante é a otimização de tempo, o planejamento prévio para a elaboração das etapas do projeto é importante para não deixar que fiquem muitas alterações e implementações dentro da API a serem realizadas em um curto período de tempo.

Melhorias futuras também serão implementadas ao projeto, como por exemplo a integração a um banco de dados, visto que teremos um número alto de informações a serem armazenadas e executadas.

Referências

API de Gestão Escolar. Disponível em: <https://school-system-spi.onrender.com/docs>.

DAIANA S. Flask Python: como usar e qual sua função? | Homehost. Disponível em: <https://www.homehost.com.br/blog/pythondjango/flask-python/#Aplicacao_com_3_rotas_e_padrao_REST_GET_POST_PUT_DELETE>.

“Test Coverage — Flask Documentation (3.1.x).” Palletsprojects.com, 2018, flask.palletsprojects.com/en/stable/tutorial/tests/.

POKEMAOBR, R. C. Flask Python: o que é e como funciona? Disponível em: <https://www.locaweb.com.br/blog/temas/codigo-aberto/flask-phyton-o-que-e/>.

MONTEIRO, V. N.; CARVALHO, G. A.; CARDOSO, V. Integração de tecnologias: python, arduino, redes neurais e flask para soluções tecnológicas eficientes. Apoena, v. 6, p. 224–229, 2023.

Nataniel Paiva – Medium. Disponível em: <https://nataniel-paiva.medium.com/>.

Arquiteto Cloud. Disponível em: <https://arquitetocloud.com/>.

HTTPS://PLUS.GOOGLE.COM/U/0/+DATACAMP. Learn R, Python & Data Science Online. Disponível em: <https://www.datacamp.com/>.