FACULDADE IMPACTA

GRADUAÇÃO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Projeto Flask

Beatriz Bramont Oliveira

Giovanna Petrilli Venditti

Isadora Cristyne de Lima Silva

São Paulo

10/04/2025

Faculdade IMPACTA

Relatório

3

Relatório Projeto Flask - Entrega 3

1. Descrição e Análise do Caso

Nesta etapa do projeto, foi realizada a integração da aplicação Flask com o Banco de Dados SQLite e posteriormente passamos para o MySQL, permitindo a persistência e a gestão estruturada das informações. Criamos o Swagger, para documentar e testar os endpoints da API de forma visual e interativa, facilitando a validação e o consumo dos serviços desenvolvidos.

Além disso, foi implementado o uso do Docker, possibilitando a criação de containers para padronizar o ambiente de desenvolvimento e a implantação da aplicação de forma prática e escalável.

A utilização do Docker foi essencial para a padronização do ambiente de desenvolvimento e implantação da API. Foram criadas imagens, que são responsáveis por encapsular a aplicação juntamente com todas as suas dependências, gerando containers isolados e portáteis, no qual garantem que a API seja executada de forma consistente em diferentes ambientes, eliminando problemas relacionados a incompatibilidades de sistema. Além disso, o Docker contribui significativamente para a escalabilidade e praticidade no processo de deploy, tornando o ciclo de desenvolvimento mais eficiente e confiável.

2. Implementação ou Procedimento

Etapas Realizadas:

1. **Integração com o banco SQLite e exclusão do banco como dicionário:** Passamos as informações que estavam organizadas por meio de um dicionário para o banco SQLite. Precisamos criar uma classe para cada entidade dentro dos models que realiza a criação das tabelas no banco de dados.

   Ajustamos também o config.py com as informações de conexão com o banco e fizemos as demais alterações nos arquivos.

   - **Ajustes no TDD:** Precisamos alterar os testes para passar com a nova integração com o banco de dados;

   - **Documentação com Swagger:** implementamos o Swagger (através do pacote flask-restx) para gerar uma documentação automática e interativa dos endpoints da API;

   - **Dockerização:** utilizamos o Docker para criar a imagem e o container, padronizando o ambiente e facilitando a implantação.

     Nessa etapa criamos o dockerfile, que define de forma automatizada e reproduzível todas as etapas necessárias para a construção da imagem de uma aplicação, sendo possível configurar o ambiente ideal para execução da aplicação, garantindo padronização e evitando erros decorrentes de configurações manuais.

- **Merge para a main:** Após conferir que o código estava funcionando corretamente, realizamos a merge para a Main.

- **Deploy no Render:** Este foi o último passo, onde realizamos o deploy da nossa aplicação, sendo possível qualquer um acessar a nossa API desde que siga as orientações.

  Ao adicionar “/docs”, a pessoa acessa a nossa documentação no swagger.

  Ao adicionar “/alunos”, “/turma”, “/professor”, você terá acesso às informações cadastradas em nosso banco.

2. **Integração com o banco MySQL:** configuramos a aplicação Flask para se conectar a um banco de dados MySQL, usando pacotes como pymysql e flask_sqlalchemy para gerenciar a persistência de dados.

   Utilizar o banco MySQL traz suporte a múltiplos usuários e conexões simultâneas com bom controle de concorrência, alta performance para volumes de dados e uso em produção e recursos avançados, como replicação, procedures, triggers, views.

   Alteramos os models das entidades aluno, professor e turma, para formar a tabela de forma correta no MySQL e configuramos o config para se conectar com o banco de dados que foi criado (escola).

- **Alteração do TDD + criação dos testes de Conexão com o banco de dados:** Realizamos alterações nos TDD’s já existentes para passar com a nova integração com o banco MySQL e criamos os testes de conexão para cada entidade.

- **Criação do docker compose:** Neste arquivo colocamos as informações necessárias para conexão com o banco de dados. Ele permite subir um banco de dados MySQL, garantindo reprodutibilidade, fácil integração com aplicações com a nossa aplicação, persistência de dados por volumes, isolamento para testes, elimina problemas de compatibilidade entre sistemas operacionais e reduz o tempo de setup manual;

- **Dockerização:** Após a configuração do docker compose executamos comando do docker no próprio terminal do Visual Studio Code para criar um contêiner com a nossa imagem;

- **Swagger:** Verificamos se as alterações realizadas foram feitas com sucesso através do nosso swagger;

- **Deploy:** Após o sucesso de cada etapa realizamos o deploy da nossa branch “testesBiaeIsa”, que contém nosso código configurado com o MySQL, no render.

Cada etapa foi importante para garantir um sistema funcional, documentado e fácil de portar.

Desafios e Decisões:

Principais Desafios:

- Execução 100% correta dos testes;
- Dockerização;
- Configurar o tempo de espera entre a inicialização do MySQL e o Flask no Docker.

Decisões Tomadas:

- Criar a API com o banco de dados MySQL;
- Realizar o deploy de ambas as branches (SQLite - Main / MySQL - testesBiaeIsa);
- Criação da documentação com o Swagger.

Essas escolhas facilitaram a execução da aplicação e a clareza na documentação da API.

Divisão de tarefas:

1. Líder: Beatriz Bramont Oliveira
    - Repartição dos testes, ajustes nos models, config e app.py, criação dos testes de integração, criação do banco de dados, garantir que os testes passassem após as mudanças, dockerização e relatório.
2. Isadora: Suporte na repartição dos testes, ajustes nos models, criação do swagger e relatório.
3. Giovanna: Relatório.

3. Resultados

Testes ou Observações Realizadas

Foram realizados testes nos endpoints da API utilizando o Swagger, validando operações de cadastro, consulta, atualização e remoção de dados.

O comportamento esperado era que cada requisição retornasse o código de status HTTP adequado (“200 OK” para consultas e atualizações, “201 Created” para cadastros e “204 No Content” para deleções) e que os dados fossem corretamente persistidos em ambos os Bancos de Dados.

Todos os testes retornaram os códigos esperados e confirmaram que as operações CRUD estavam funcionando corretamente. Além disso, o ambiente em Docker se mostrou estável, permitindo a execução simultânea da aplicação Flask e do banco de dados.

Os resultados atenderam às expectativas?

Os resultados obtidos atenderam plenamente às expectativas iniciais, demonstrando que a integração entre o Flask, SQLite, MySQL e o Docker foram bem-sucedidas.

O funcionamento correto dos endpoints confirma que a API foi construída de forma consistente e a utilização do Swagger facilitou a validação do comportamento da aplicação.

O uso de containers Docker revelou-se eficaz para garantir a padronização e a facilidade de execução em qualquer ambiente, o que reforça a importância da conteinerização no desenvolvimento de aplicações modernas.

4. Conclusão

O projeto permitiu entender como integrar uma aplicação web utilizando Flask com um banco de dados (tanto SQLite quanto MySQL), documentar APIs com Swagger, gerenciar todo o ambiente usando Docker e como realizar o deploy de uma aplicação.

Essas práticas mostraram que é possível criar sistemas mais organizados, portáteis e fáceis de testar, alinhando-se ao objetivo inicial de aprender a construir e disponibilizar APIs completas e funcionais.

Para ampliar e otimizar o projeto, podem ser implementadas melhorias como:

- Adicionar autenticação de usuários para proteger os endpoints;
- Implantar a aplicação em ambientes de nuvem (AWS, Heroku);
- Expandir a API para incluir relacionamentos mais complexos entre tabelas no banco de dados.

5. Impacto e Conexão com o Mundo Real

A construção de APIs utilizando Flask integradas com banco de dados MySQL e SQLite, documentadas via Swagger e executadas em containers Docker tem aplicação direta em diversos cenários profissionais.

Essas tecnologias são amplamente usadas em sistemas de comércio eletrônico, plataformas de serviços financeiros, aplicações de logística e gerenciamento de dados, entre muitos outros.

Impacto Estrutural e Operacional:

Centralização e Organização: A implementação de uma API com Flask estruturada, banco de dados SQLite e MySQL, documentação Swagger e gerenciamento por Docker proporciona uma centralização eficiente dos dados e serviços. Essa organização facilita a manutenção do sistema, o desenvolvimento em equipe e a rápida evolução de novas funcionalidades.

Conexão com o Mundo Real: Empresas podem estruturar seus sistemas de atendimento ao cliente, vendas online ou controle interno de forma modular, facilitando atualizações rápidas e adaptações a novos requisitos de mercado, sem prejudicar a operação contínua.

Impacto na Experiência do Usuário:

Capacidade de Expansão: O uso de containers Docker e banco de dados relacional permite que a aplicação seja facilmente escalada para atender maior volume de usuários ou novos serviços, além de possibilitar integrações com outros sistemas ou APIs externas.

Conexão com o Mundo Real: Negócios de e-commerce, plataformas de saúde ou gestão de transportes podem ampliar suas operações integrando novos módulos ou serviços sem precisar reescrever toda a infraestrutura.

6. Desafios Futuros e Melhorias

Durante o desenvolvimento do projeto, enfrentamos desafios como a instabilidade na conexão entre Flask e MySQL no ambiente Docker, que pode ser solucionada futuramente com ajustes no tempo de inicialização dos serviços e reconexão automática. Como melhorias, sugerimos a implementação de autenticação JWT para proteger as rotas, a criação de testes automatizados para validar as APIs e a publicação da aplicação em plataformas de nuvem como AWS.

O aprendizado pode ser expandido para projetos reais, criando sistemas seguros, documentados e eficientes, aplicáveis em diversas áreas do mercado.

Referências:

Swagger documentation — Flask-RESTX 1.1.1.dev documentation. Disponível em: <https://flask-restx.readthedocs.io/en/latest/swagger.html>.

OpenAPI Specification - Version 3.0.3 | Swagger. Disponível em: <https://swagger.io/specification/>.

TERRALAB. Documentando sua API Rest com Swagger - TerraLAB - Medium. Disponível em: <https://medium.com/@terralab/documentando-sua-api-rest-com-swagger-4c39e92ddd12>.

Medium. Disponível em: <https://medium.com/@kattsonbastos/docker-deploy-de-modelos-e-ci>.

Medium. Disponível em: <https://medium.com/@raphael.melo_93066/criando-sua-aplica>.