Relatório de Testes de API - Casos de Teste CRUD em Ambiente Swagger (FakeRESTApi)

Jordan S. de Souza Pinheiro[1], Maria Eduarda V. Oliveira[2], Liandra Silva da Silva[3], Lucas Giovanini dos S. Minervino[4], Gabriel Valente de Oliveira[5]

[1] Faculdade Metropolitana de Manaus (FAMETRO)

Jordansspinheiro@gmail.com, m.eduardavlt@gmail.com, gabriel.valente.567@gmail.com, lucasaqw7@hotmail.com, liasilv099@gmail.com

Abstract. *This paper describes the individual contribution to the API Testing Project, focusing on the five essential CRUD (Create, Read, Update, Delete) test cases. The methodology used the Pytest framework in Python and targeted the FakeRESTApi, which provides a Swagger (OpenAPI) interface for verification. The goal was to validate API responsiveness, expected status codes (200 OK), and data manipulation for all major HTTP methods: POST, GET, PUT, and DELETE.*

Resumo. *Este artigo descreve a contribuição individual para o Projeto de Testes de API, focando nos cinco casos de teste essenciais do CRUD (Criar, Ler, Atualizar, Deletar). A metodologia utilizou o framework Pytest em Python e foi direcionada à FakeRESTApi, que fornece uma interface Swagger (OpenAPI) para validação. O objetivo foi validar a resposta da API, códigos de status esperados (200 OK) e manipulação de dados para todos os principais métodos HTTP: POST, GET, PUT e DELETE.*

1. Introdução

Este artigo apresenta contribuições individuais para um projeto de teste de API focado na validação de operações CRUD (Criar, Ler, Atualizar, Deletar) aplicadas a recursos expostos pela FakeRESTApi.

O projeto em grupo estabelece um requisito mínimo para o uso de uma API simulada com pelo menos 30 endpoints, garantindo uma estrutura robusta que abranja diversas entidades, como Usuários, Livros e Atividades. A metodologia utilizada visa validar o desempenho do backend por meio da execução de testes automatizados utilizando o framework Pytest.

2. Metodologia de Testes e Resultados Individuais (CRUD)

O trabalho proposto envolve a implementação de cinco casos de teste que abrangem quatro operações CRUD básicas aplicadas a recursos de API: /Activities, /Authors, /Books, /Users e /CoverPhotos: https://fakerestapi.azurewebsites.net/api/

Para garantir a ordem e a rastreabilidade, o código de teste foi estruturado de acordo com os métodos HTTP utilizados (POST, GET, PUT e DELETE). Em todos os

cenários, o comportamento esperado do FakeRESTApi é retornar um código de status 200 OK, indicando que a operação solicitada foi processada com sucesso, conforme o comportamento padrão de simulação.

2.1. Maria Eduarda Valente (/Activities)

- **Teste POST (Create)**

  Explicação: Este teste verifica a capacidade de **criação** de um novo recurso no servidor. Enviamos um *payload* com dados de atividade para o *endpoint* /Activities. O código de *status* esperado é 200 OK. No contexto desta API de simulação, o objeto retornado reflete os dados enviados, sendo o ID retornado como 0.

Payload Utilizado (tests/test_post.py**):**

```
{
  "id": 0,
  "title": "Atividade Criada para o Trabalho",
  "dueDate": "2025-12-01T10:00:00.000Z",
  "completed": false
}
```

- **Teste GET (Read)**

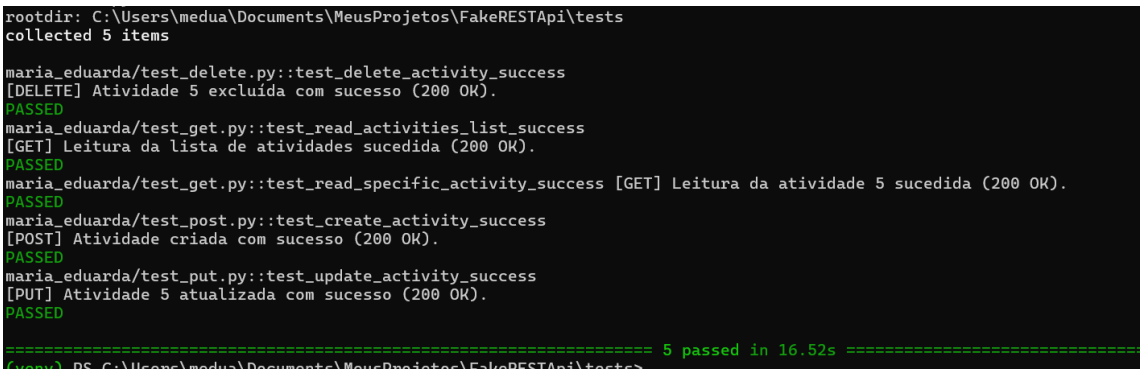
  Explicação: Foram realizados dois testes para a operação **Leitura**: obter a lista completa de atividades (GET /Activities) e obter uma atividade específica (GET /Activities/{id}). Ambos esperam o Status Code **200 OK** e a presença dos dados no corpo da resposta (JSON).

- **Teste PUT (Update)**

  Explicação: O teste **PUT** verifica a **atualização** completa de um recurso, alterando o título e o *status* completed da atividade de ID 5. O *payload* inclui todos os campos do recurso. O teste espera o Status Code **200 OK** e verifica se o novo título foi retornado na resposta.

- **Teste DELETE (Delete)**

  Explicação: O teste **DELETE** verifica a **exclusão** de um recurso existente, enviando a requisição para /Activities/5. O Status Code esperado é **200 OK**, e o teste verifica se o corpo da resposta está vazio, confirmando a exclusão simulada.



2.2. Jordan S de Souza Pinheiro (/Activities)

- **Teste POST (Create)**
  Explicação: utilizei o método **POST** para validar se a **API** é capaz de registrar uma nova atividade no endpoint /Activities. O envio do payload cria um objeto completo, e o teste confirma se a **API** retorna exatamente o recurso informado. O status **200 OK** reforça que o servidor aceitou e processou corretamente a requisição.

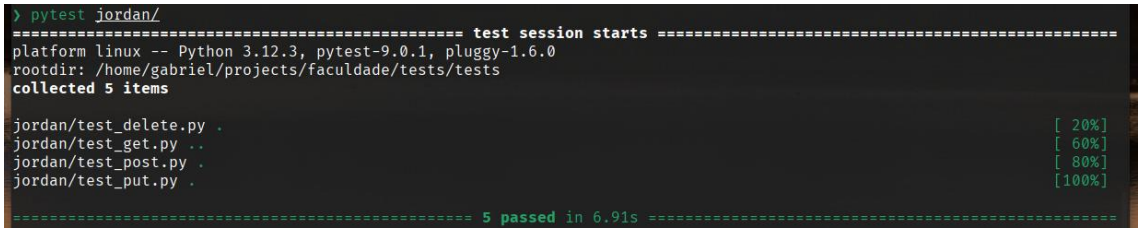
Payload Utilizado (tests/test_post_jordan.py):

```
{
  "id": 10,
  "title": "Atividade Criada por Jordan",
  "dueDate": "2025-10-10T00:00:00.000Z",
  "completed": true
}
```

- **Teste GET (Read)**
  Explicação: Para validar a leitura, verifiquei tanto a listagem geral de atividades quanto a recuperação de uma atividade específica. O objetivo foi confirmar a consistência dos dados retornados pela **API**, que respondeu com **200 OK** e JSON bem estruturado em todas as consultas.

- **Teste PUT (Update)**
  Explicação: O teste de atualização substitui integralmente os dados da atividade de ID 10. avaliei se a **API** realmente sobrescreve os valores anteriores e devolve o recurso modificado. O retorno **200 OK** e o novo título no JSON confirmam o sucesso da operação.

- **Teste DELETE (Delete)**
  Explicação: avaliei o comportamento da remoção de recursos. Ao solicitar o **DELETE** em /Activities/10, a **API** retornou **200 OK** e uma resposta vazia, o que demonstra que o recurso foi simulado como excluído pelo servidor.



2.3. Gabriel Valente de Oliveira (/Activities)

- **Teste POST (Create)**
  Explicação: validei a criação de uma atividade enviando um objeto completo para /Activities. Esse teste assegura que a **API** aceita novos dados e devolve o mesmo conteúdo enviado, seguindo o padrão desta **API** fake. A resposta **200 OK** indica processamento bem-sucedido.

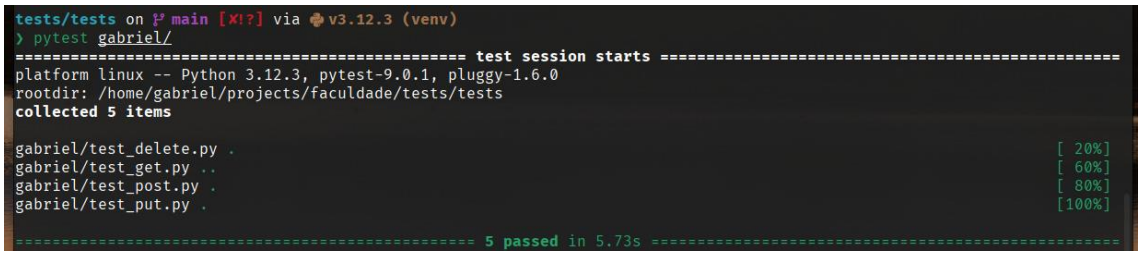
Payload Utilizado (tests/test_post_gabriel.py):

```
{
  "id": 20,
  "title": "Atividade Criada por Gabriel",
  "dueDate": "2024-08-10T15:00:00.000Z",
  "completed": false
}
```

- **Teste GET (Read):**
  Explicação: verifiquei se o endpoint de leitura funciona corretamente tanto para retornar todas as atividades quanto para localizar uma específica pelo ID. Ambos retornaram **200 OK**, com dados coerentes e formato JSON adequado.

- **Teste PUT (Update)**
  Explicação: Este teste garante que a **API** consegue atualizar integralmente o recurso /Activities/20. alterei o título e o status de conclusão. A resposta **200 OK** e a confirmação dos novos dados no corpo demonstram que a operação foi aplicada.

- **Teste DELETE (Delete)**
  Explicação: O teste **DELETE** tem como objetivo verificar se o servidor responde adequadamente ao solicitar a remoção de /Activities/20. A resposta vazia acompanhada do status **200 OK** confirma a simulação da exclusão.



2.4. Liandra Silva da Silva (/Activities)

- **Teste POST (Create)**
  Explicação: Testei a inclusão de uma nova atividade enviando um payload completo para o endpoint /Activities. O teste avalia se o servidor replica os dados corretamente e se aceita o registro. A resposta **200 OK** garante que o recurso foi processado com sucesso.

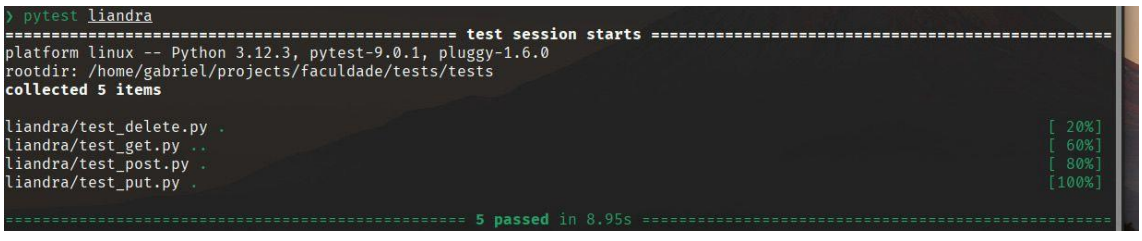
Payload Utilizado (tests/test_post_liandra.py):

```
{
  "id": 15,
  "title": "Atividade Criada por Liandra",
  "dueDate": "2024-11-05T09:30:00.000Z",
  "completed": true
}
```

- **Teste GET (Read)**
  Explicação: Para assegurar a confiabilidade da leitura, consultei tanto a lista de atividades quanto uma atividade específica. O servidor respondeu com **200 OK**, retornando dados completos e coerentes com o esperado.

- **Teste PUT (Update)**
  Explicação: O teste **PUT** atualiza todos os campos do recurso /Activities/15. O foco foi verificar se a API realmente substitui o conteúdo anterior. A resposta **200 OK** e o novo título confirmam que o sistema aceitou a modificação.

- **Teste DELETE (Delete)**
  Explicação: avaliei o endpoint **DELETE** solicitando a exclusão do recurso de ID 15. O teste foi aprovado ao receber **200 OK** com corpo vazio, o comportamento típico desta **API** simulada.



2.5. Lucas Giovanini dos S Minervino (/Activities)

- **Teste POST (Create)**
  Explicação: enviei um payload ao endpoint /Activities para verificar se a **API** aceita a criação de novos recursos. O retorno deve refletir os dados cadastrados e responder com **200 OK**, indicando que o servidor processou corretamente a requisição.

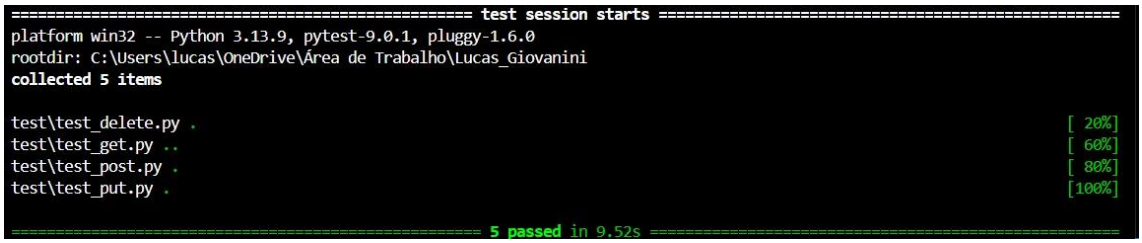
Payload Utilizado (tests/test_post_lucas.py):

```
{
  "id": 25,
  "title": "Atividade Criada por Lucas",
  "dueDate": "2026-03-22T12:00:00.000Z",
  "completed": false
}
```

- **Teste GET (Read)**
  Explicação: testei a funcionalidade de leitura da **API**, tanto listando todas as atividades quanto consultando uma específica pelo ID. Em ambos os casos, o endpoint respondeu com **200 OK** e dados JSON consistentes.

- **Teste PUT (Update)**
  Explicação: No teste de atualização, alterei as informações completas do recurso /Activities/25. O teste valida se o servidor retorna o recurso atualizado e finaliza com o status **200 OK**, garantindo que a operação funcionou como esperado.

- **Teste DELETE (Delete)**
  Explicação: Ao enviar a requisição **DELETE** para /Activities/25, avaliei se a **API** simula a remoção corretamente. O status **200 OK** e o corpo vazio indicam que a exclusão foi concluída.



3. Escopo do Projeto de Grupo

O projeto em grupo utilizou a API FakeRESTApi como base para o desenvolvimento, pois ela atende aos requisitos mínimos de 30 endpoints distintos. Essa API fornece uma variedade significativa de recursos, incluindo endpoints como /Activities, /Books, /Users, /Authors, /CoverPhotos e caminhos de autenticação específicos em /Auth. A existência dessas múltiplas entidades permite um cenário de teste amplo e robusto.

Essa variedade de recursos suporta a validação detalhada dos métodos HTTP (GET, POST, PUT, DELETE), que são fundamentais para as operações CRUD (Criar, Ler, Atualizar, Excluir). Isso permite a criação de múltiplos cenários de teste para garantir a funcionalidade e a integridade dos dados simulados, garantindo uma avaliação funcional completa do sistema.

Utilizando dados simulados e uma variedade de operações, torna-se fácil aprender e desenvolver habilidades fundamentais em testes de software e integração de APIs.

4. References

1. FakeRESTApi – Documentação Oficial
FAKERESTAPI. *Fake REST API for testing*. Disponível em: https://fakerestapi.azurewebsites.net/index.html.
Acesso em: 26 nov. 2025.

2. Swagger / OpenAPI Specification
OPENAPI Initiative. *OpenAPI Specification v3.0*. Disponível em: https://swagger.io/specification/.
Acesso em: 26 nov. 2025.

3. Pytest – Documentação Oficial
PYTEST. *Pytest Documentation*. Disponível em: https://docs.pytest.org/.
Acesso em: 26 nov. 2025.

4. Requests – Biblioteca Python para Requisições HTTP
REQUESTS. *Requests: HTTP for Humans*. Disponível em: https://requests.readthedocs.io/.
Acesso em: 26 nov. 2025.

5. GitHub – Repositório do Projeto
DWDAA. *FakeRESTApi – GitHub Repository*. Disponível em: https://github.com/dwdaa/FakeRESTApi/.
Acesso em: 26 nov. 2025.