**Implementando API de Imagens com FastAPI**

**Transferência de Estilo com Flutter, FastAPI e TensorFlow**

Este é o quinto post da série Transferência de Estilo com Flutter, FastAPI e TensorFlow. Nesta série, estou documentando minha experiência desenvolvendo uma aplicação de Machine Learning (ML) do início ao fim (end-to-end).

* [Parte 1 - Transferência de Estilo com Flutter e TensorFlow](https://matheper.com/2020/07/27/transfer%C3%AAncia-de-estilo-com-flutter-e-tensorflow/)
* [Parte 2 - Primeiros Passos com Flutter](https://matheper.com/2020/08/31/transferencia-de-estilo-com-flutter-e-tensorflow-2/)
* [Parte 3 - Flutter e suporte ao TensorFlow Lite](https://matheper.com/2020/09/07/flutter-e-suporte-ao-tensorflow-lite/)
* [Parte 4 - Machine Learning no Device ou Servidor](https://matheper.com/2020/09/12/machine-learning-no-device-ou-servidor/)

Acabei sendo mais rápido para implementar a aplicação do que para publicar no blog. Se você quiser, já pode acessar o [código completo no GitHub](https://github.com/matheper/style_transfer).

## Introdução

Utilizar FastAPI não estava no planejamento inicial do projeto mas, devido a [falta de suporte nativo ao TensorFlow Lite no Flutter](https://matheper.com/2020/09/07/flutter-e-suporte-ao-tensorflow-lite/), decidi [mover nosso modelo de Machine Learning do device para um servidor backend](https://matheper.com/2020/09/12/machine-learning-no-device-ou-servidor/).

Vamos utilizar FastAPI para implementar o backend já que, desta forma, podemos executar nosso modelo diretamente no ambiente Python e disponilizá-lo como um simples request de uma API REST.

## FastAPI

FastAPI é um framework Python que promete entregar aplicações de alta performance e prontas para produção com uma síntaxe facil de aprender e rápido de implementar.

FastAPI framework, high performance, easy to learn, fast to code, ready for production (fastapi.tiangolo.com)

Duas características me surpreenderam logo que comecei meus testes: design e performance. Vou focar nestas duas, mas você pode encontrar a [lista completa de features](https://fastapi.tiangolo.com/features) diretamente na documentação.

### Design

O que mais me chamou atenção quanto ao design está relacionado ao sistema de [anotações do Python (Type Hints)](https://docs.python.org/3/library/typing.html) e a biblioteca [Pydantic](https://pydantic-docs.helpmanual.io/). A partir das anotações Python, FastAPI faz a validação da API e a conversão automática dos seus dados de entrada e saída.

Isso significa que você pode receber um JSON complexo e ele será convertido automaticamente em um objeto Python, com uso de [modelos Pydantic](https://fastapi.tiangolo.com/python-types/#pydantic-models), respeitando tipos como: str, int, float, list, datetime, etc.

O inverso também é válido, você retorna um objeto Python complexo em seu código e FastAPI fará a conversão para um "tipo web" como [HTML](https://fastapi.tiangolo.com/advanced/custom-response/#html-response), [JSON](https://fastapi.tiangolo.com/advanced/response-directly/#using-the-jsonable_encoder-in-a-response), [XML](https://fastapi.tiangolo.com/advanced/response-directly/#returning-a-custom-response), [Plain Text](https://fastapi.tiangolo.com/advanced/custom-response/" \l "plaintextresponse), [File](https://fastapi.tiangolo.com/advanced/custom-response/#plaintextresponse), [Streaming](https://fastapi.tiangolo.com/advanced/custom-response/#streamingresponse), [etc](https://fastapi.tiangolo.com/advanced/custom-response/#custom-response-html-stream-file-others).

Outra característica sensacional é a geração automática de documentação de padrão aberto [OpenAPI](https://github.com/OAI/OpenAPI-Specification) (anteriormente conhecida como [Swagger](https://swagger.io/)) e [JSON Schema](http://json-schema.org/). Sem nenhum código adicional, você tem uma documentação completa e que permite testes de forma interativa.

Uma última vantagem de FastAPI e a imposição de Type Hints com Pydantic, é que [sistemas de auto-completar de editores como Pycharm e VSCode sempre funcionam](https://fastapi.tiangolo.com/features/#editor-support). Sua IDE será capaz de fornecer informações como assinaturas de metódos, atributos ou tipos de dados enquanto você está programando.

### Performance

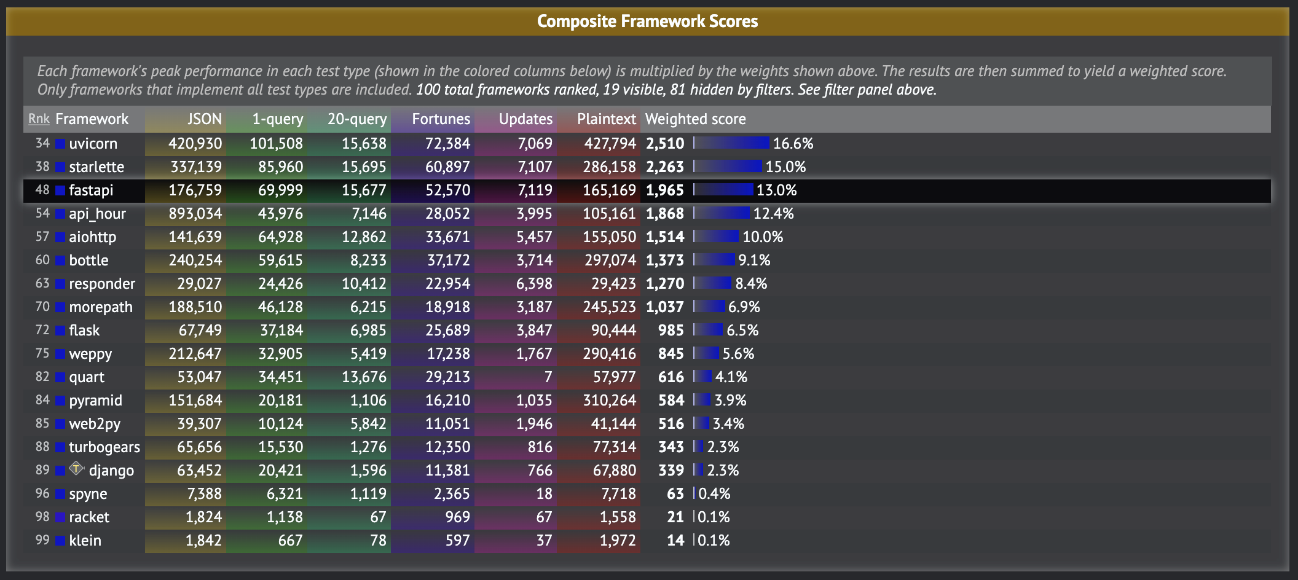
A alta performance está diretamente relacionada às bibliotecas [Starlette](https://www.starlette.io/) e [Uvicorn](https://www.uvicorn.org/), que FastAPI utiliza internamente.

Uvicorn é um Servidor Python [ASGI - Asynchronous Server Gateway Interface](https://asgi.readthedocs.io/en/latest/) (sucessor do [WSGI](https://wsgi.readthedocs.io/en/latest/what.html)) que utiliza [uvloop](https://github.com/MagicStack/uvloop) para a execução de aplicações assíncronas.

Starlette é um framework baseado em Uvicorn que fornece algumas abstrações úteis para o desenvolvimento web como rotas, responses, events, CORS, Session, Cookie, etc. Além disso, Starlette suporta [Web Sockets](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API) e [GraphQL](https://graphql.org/).

FastAPI é uma terceira camada de abstração implementada por cima de Starlette e Uvicorn. Ele tem suporte a todas as features anteriores, mais abstrações e features adicionais como validação de dados e documentação automática.

Em benchmarks, FastAPI fica em terceiro lugar em relação à performance de libs Python para web, atrás apenas das próprias libs Uvicorn e Starlette. Entretanto, é preciso lembrar que estas três bibliotecas fornecem níveis diferentes de abstrações e facilidades para o desenvolvimento web.

 (fonte: [TechEmpower Web Framework Benchmarks](https://www.techempower.com/benchmarks/" \l "section=data-r18&hw=ph&test=composite&l=zijzen-7&a=2))

Utilizar apenas Uvicorn trará o máximo de performance em Python para web, mas o mínimo de features. Starlette será um meio termo entre performance e features. FastAPI será mais completo em features, porém menos performático.

Performance: Uvicorn > Starlette > FastAPI

Abstrações: FastAPI > Starlette > Uvicorn

Resumindo, quanto mais especializada uma biblioteca, mais rápida ela é, quanto mais abrangente, menos performática. Uvicorn, Starlette e FastAPI estão em uma hierarquia e não competindo entre si. Starlette estende as features de Uvicorn e FastAPI estende as features de Starlette (e Uvicorn).

FastAPI adiciona um nível excelente de abstrações ao custo de alguma performance (comparado a Uvicorn e Starlette), ainda assim, sendo mais rápido que outros frameworks Python como [AIOHTTP](https://docs.aiohttp.org/en/stable/), [Bottle](https://bottlepy.org/docs/dev/), [Flask](https://flask.palletsprojects.com/) e [Django](https://www.djangoproject.com/).

## Instalação

Antes de começar com nossa implementação, vamos preparar o ambiente de desenvolvimento. É bem simples instalar FastAPI, você só precisa ter o [Python](https://www.python.org/) instalado e do comando [pip](https://pypi.org/project/pip/).

Algumas distribuições Linux já vêm com Python por padrão, se este não for seu caso, você pode [baixar o instalador](https://www.python.org/downloads/) diretamente do site oficial.

Antes de instalar qualquer biblioteca no seu sistema, recomendo criar um ambiente virtual com [venv](https://docs.python.org/3/library/venv.html), [conda](https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#creating-an-environment-with-commands), [pipenv](https://pypi.org/project/pipenv/) ou [poetry](https://python-poetry.org/), assim você não corre o risco de bagunçar sua instalação Python principal.

Então vamos começar criando uma pasta para nosso projeto.

mkdir style\_transfer

cd style\_transfer

E depois criar e ativar um ambiente virtual. Eu estou usando venv, mas você pode usar o que achar mais conveniente.

python3 -m venv env

**source** env/bin/activate

Se os comandos deram certo, você terá uma pasta style\_transfer e dentro outra pasta chamada env. A pasta env é onde está sua nova instalação Python, criada especialmente para este projeto.

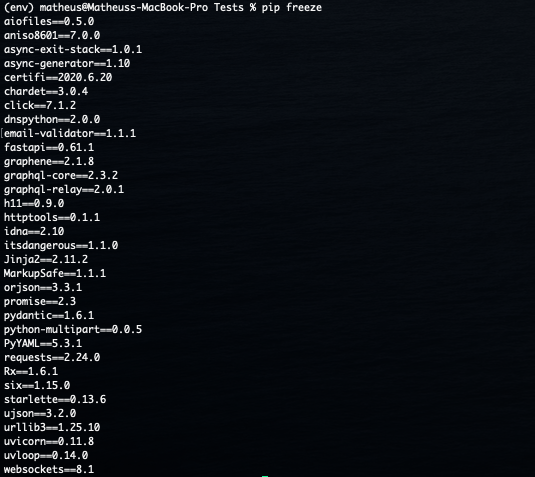
Sua linha de comando também deve estar diferente, iniciando com (env). Rodando o comando pip freeze, você verá que sua instalação Python está complementamente limpa, não haverá nenhuma biblioteca listada.

pip **freeze**

Com o ambiente virtual pronto, já podemos [instalar FastAPI](https://fastapi.tiangolo.com/tutorial/#install-fastapi). É possível selecionar quais dependências serão instaladas, mas para simplificar as instruções vamos instalar todas elas com o seletor [all].

pip install "fastapi[all]"

Se você rodar o comando pip freeze novamente, agora terá uma lista de pacotes instalados, entre eles uvicorn, starlette e pydantic (que citei anteriormente), mais alguns pacotes bem populares como urllib3, requests, PyYAML, etc.



## REST API

Com FastAPI instalado, nosso objetivo é desenvolver uma API que receba e retorne imagens. Assim, estaremos prontos para integrar nosso modelo TensorFlow nas próximas etapas.

Quando estamos implementando algo novo, é importante resolvermos um problema de cada vez, de preferência em etapas incrementais, que podem ser testadas e nos darão a sensação de progresso.

Vamos começar criando um arquivo chamado main.py que irá conter toda a lógica relacionada à nossa API.

touch main.py

Com o arquivo criado, vamos implementar nossos dois métodos: GET e POST.

### GET status

Podemos começar programando um método GET que retorne uma mensagem qualquer. Com um editor de texto de sua preferência, adicione as seguintes linhas de código ao arquivo main.py:

**from** **fastapi** **import** FastAPI

app = FastAPI()

@app.get('/')

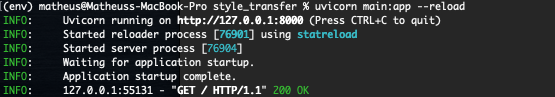
async **def** root():

**return** {'status': 'ok'}

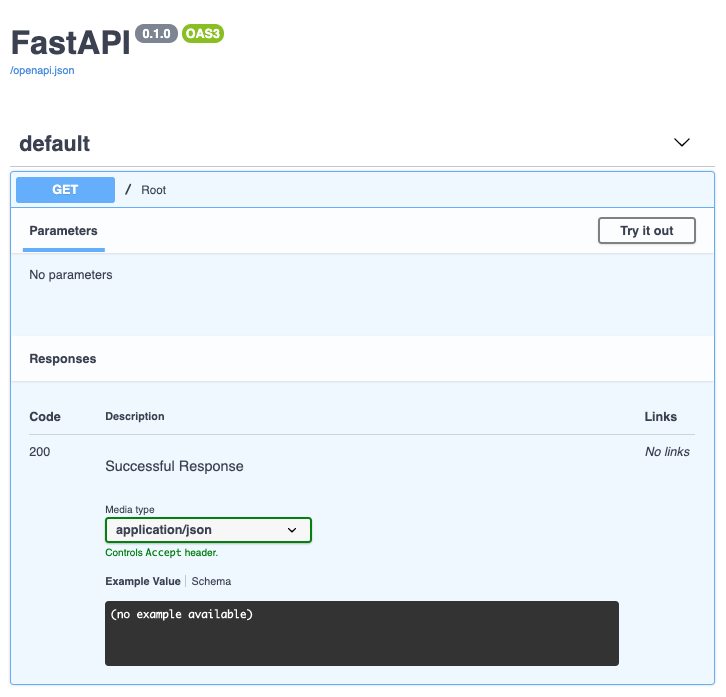
Explicando o código, importamos FastAPI e criamos nosso app. Após, registramos a raiz do nosso projeto '/' à função root, que não tem parâmetros e retorna um dicionário com status ok.

Somente com essas linhas, já temos uma aplicação que retorna um JSON em um endpoint GET. Podemos testar o código servindo o app com uvicorn:

uvicorn main:app *--reload*



Você encontrará a mensagem de status acessando http://127.0.0.1:8000/. Também é possível acessar a documentação OpenAPI e testar seu backend de forma interativa diretamente em http://127.0.0.1:8000/docs.



A [especificação OpenAPI](http://spec.openapis.org/oas/v3.0.3) também pode ser utilizada para a [geração automática de código dos clientes do seu backend](https://github.com/OpenAPITools/openapi-generator#overview).

### POST Image

Agora vamos implementar um método POST que receba uma imagem do cliente e a devolve logo em seguida, sem manter nada no servidor.

Este é o endpoint que utilizaremos para nossa transferência de estilo, então já vamos chamá-lo /style.

Vamos receber a [imagem como um arquivo](https://fastapi.tiangolo.com/tutorial/request-files) e [devolvê-la como um Streaming](https://fastapi.tiangolo.com/advanced/custom-response/#streamingresponse). Também vamos utilizar io.BytesIO para manter nossa imagem em memória.

**import** **io**

**from** **fastapi** **import** FastAPI, File

**from** **starlette.responses** **import** StreamingResponse

app = FastAPI()

@app.get('/')

async **def** root():

**return** {'status': 'ok'}

@app.post('/style')

async **def** predict(img\_bytes: bytes = File(...)):

img = io.BytesIO(img\_bytes)

img.seek(0)

**return** StreamingResponse(

img,

media\_type="image/jpg",

)

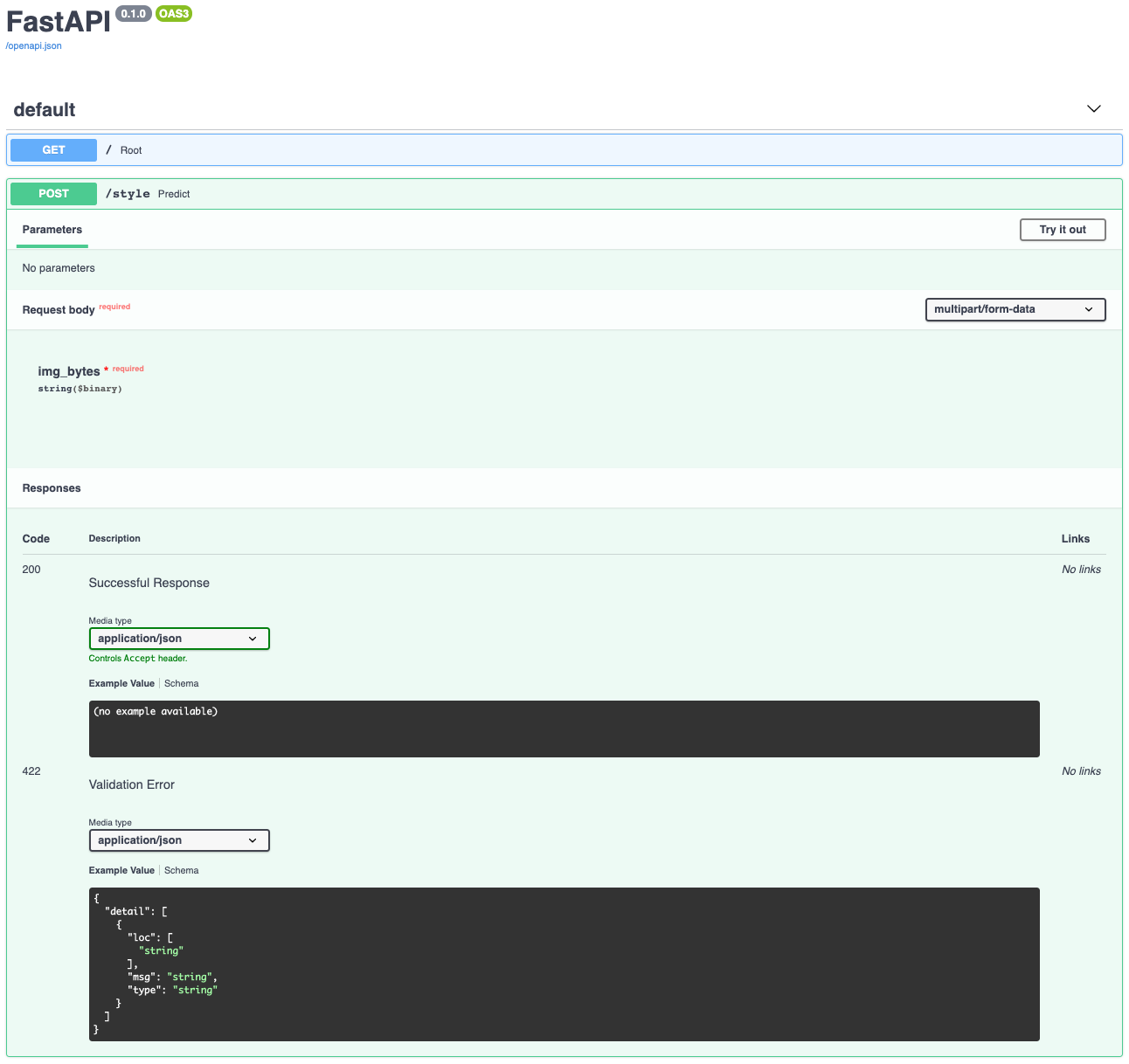
Explicando o código, importamos File e Streaming e registramos o caminho /style à função predict. Isso significa que toda vez que alguém acessar a URL http://localhost:8000/style, a função predict será invocada.

Desta vez, a função recebe um conteúdo em bytes que é armazenado na variável img\_bytes. Com o uso da anotação Python File(...), dizemos para o FastAPI que estes bytes devem vir do [upload de um arquivo](https://fastapi.tiangolo.com/tutorial/request-files/#request-files).

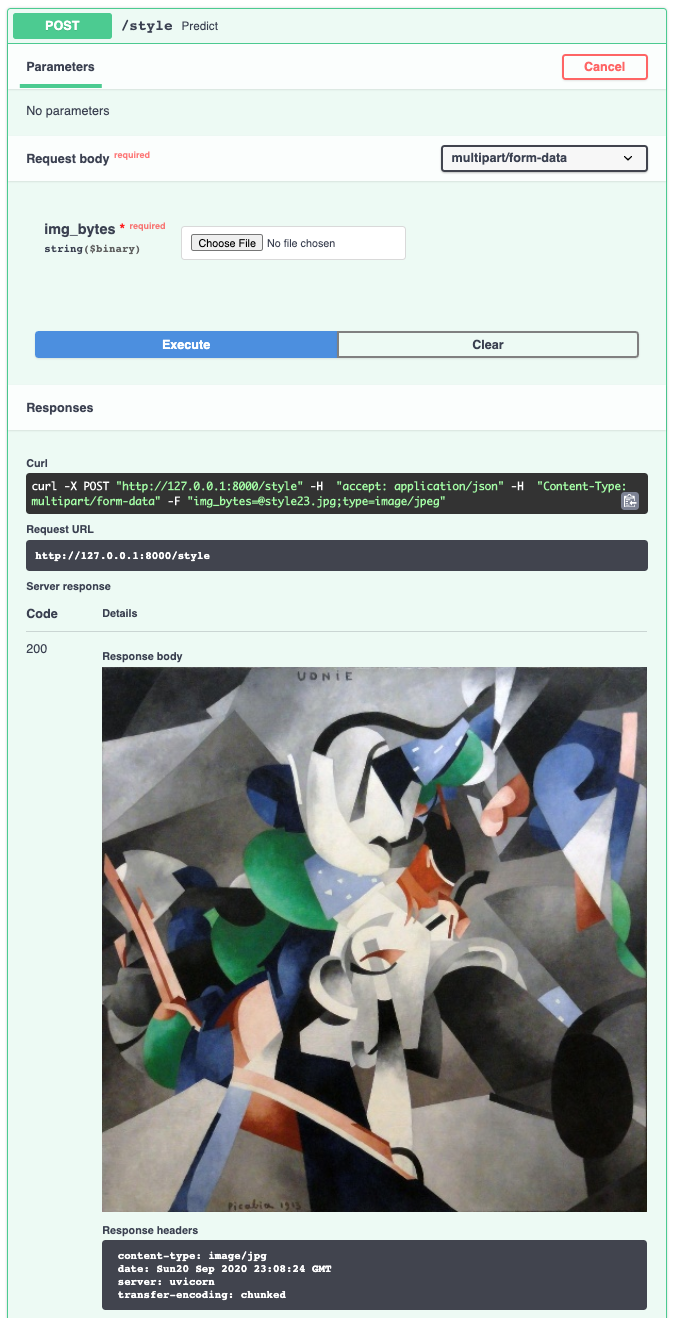
Na primeira linha da função, os bytes são carregados em memória. Logo em seguida, o ponteiro de leitura e escrita é enviado de volta ao início do lista de bytes img.seek(0). Precisamos fazer isso pois o StreamingResponse irá iterar novamente por essa lista.

Finalmente, a imagem é enviada de volta ao cliente. Isto é feito retornando o conteúdo em bytes da variável img utilizando StreamingResponse e especificando o retorno como image/jpg.

Se voltarmos à documentação, agora temos dois endpoints: GET e POST. O método POST está associado à URL /style e recebe um parâmetro img\_bytes do tipo bytes, assim como acabamos de implementar em nosso código.



Ao clicar em Try it out, é possível fazer o upload de uma imagem e, ao enviar a request para o servidor, o resultado é a própria imagem, com status 200 e content-type image/jpg.



Basicamente, esta é toda a implementação que precisamos para nossa API, além de alguns testes unitários que acabamos não escrevendo no tutorial.

**Instalar biblioteca cv2 para tratamento de imagem**

Primeiro precisamos atualizar o ‘pip’ para uma versão suportada com o comando:

python -m pip install --upgrade pip

fazer a instalação do Open CV

pip install opencv-python

fazer a instalação da biblioteca Spicy

pip install -U scipy

SciPy é uma coleção de algoritmos matemáticos e funções de conveniência construídos na extensão NumPy do Python. Ele adiciona um poder significativo à sessão Python interativa, fornecendo ao usuário comandos e classes de alto nível para manipulação e visualização de dados. Com o SciPy, uma sessão Python interativa se torna um ambiente de processamento de dados e prototipagem de sistema rival de sistemas, como MATLAB, IDL, Octave, R-Lab e SciLab.

O benefício adicional de basear o SciPy em Python é que isso também disponibiliza uma linguagem de programação poderosa para uso no desenvolvimento de programas sofisticados e aplicativos especializados. Os aplicativos científicos que usam o SciPy se beneficiam do desenvolvimento de módulos adicionais em vários nichos do cenário de software por desenvolvedores em todo o mundo. Tudo, desde a programação paralela até a web e sub-rotinas e classes de banco de dados, foram disponibilizados para o programador Python. Todo esse poder está disponível além das bibliotecas matemáticas no SciPy.

“Utilizaremos essa biblioteca para utilizar alguns algoritmos matemáticos para aplicar efeitos em imagens”.

Vamos agora instalar as dependências necessárias para esta etapa da implementação: [TensorFlow](https://www.tensorflow.org/install) para interagir com os modelos de Machine Learning e [Pillow](https://pillow.readthedocs.io/en/stable/installation.html" \l "basic-installation) para nos ajudar com a manipulação das imagens.

pip install -U pillow

A Python Imaging Library adiciona recursos de processamento de imagem ao seu interpretador Python.

Essa biblioteca oferece amplo suporte a formatos de arquivo, uma representação interna eficiente e recursos de processamento de imagem bastante poderosos.

A biblioteca de imagens centrais é projetada para acesso rápido aos dados armazenados em alguns formatos básicos de pixel. Deve fornecer uma base sólida para uma ferramenta geral de processamento de imagem.

pip install tensorflow

explicações sobre o Tensorflow:

https://www.tensorflow.org/?hl=pt-br