FastAPI

**Agenda**

* Introduction
* Features
* Installation
* Usage
* FastAPI vs Flask
* Cons
* Conclusion
* Reference

Image

# Introduction

FastAPI is a modern and as the name is says fast web framework to build APIs in Python.

Author of this framework is Sebastián Ramírez. Available from Jan 2019 (based on release notes).

# Features

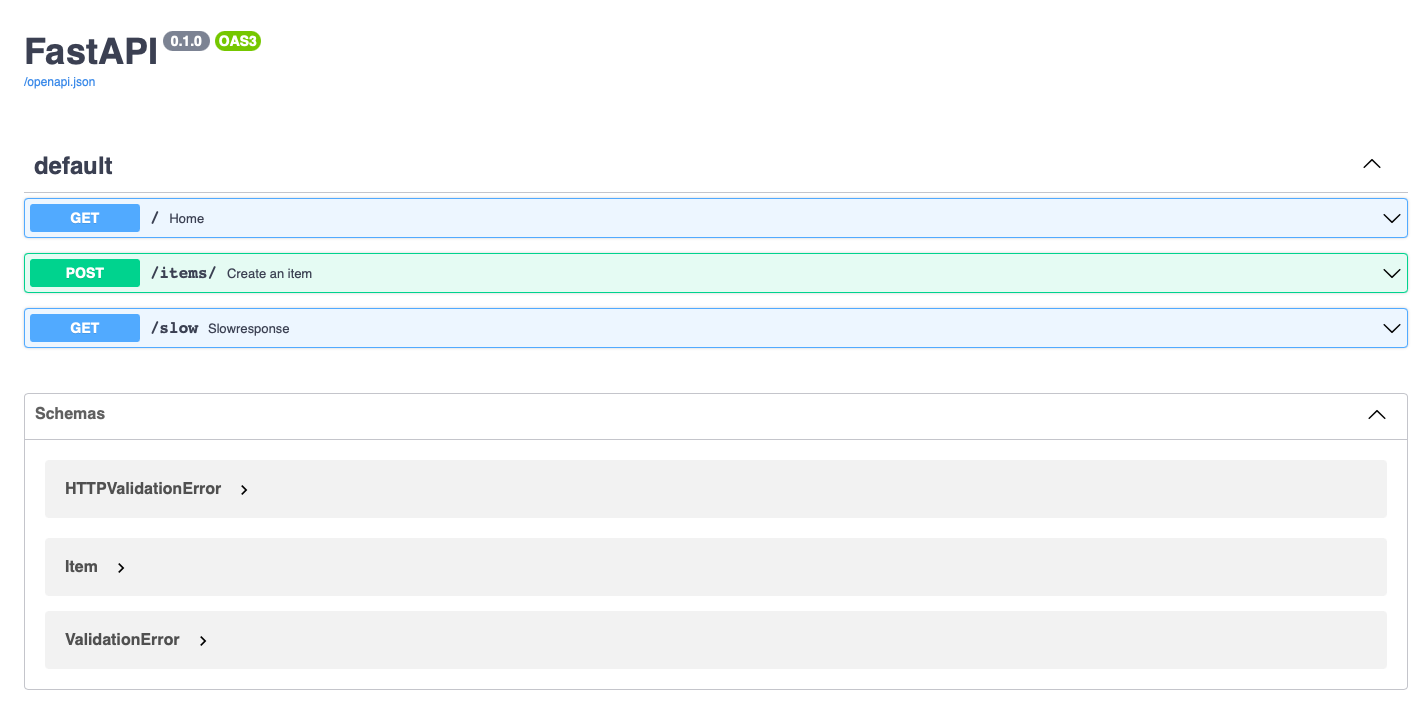
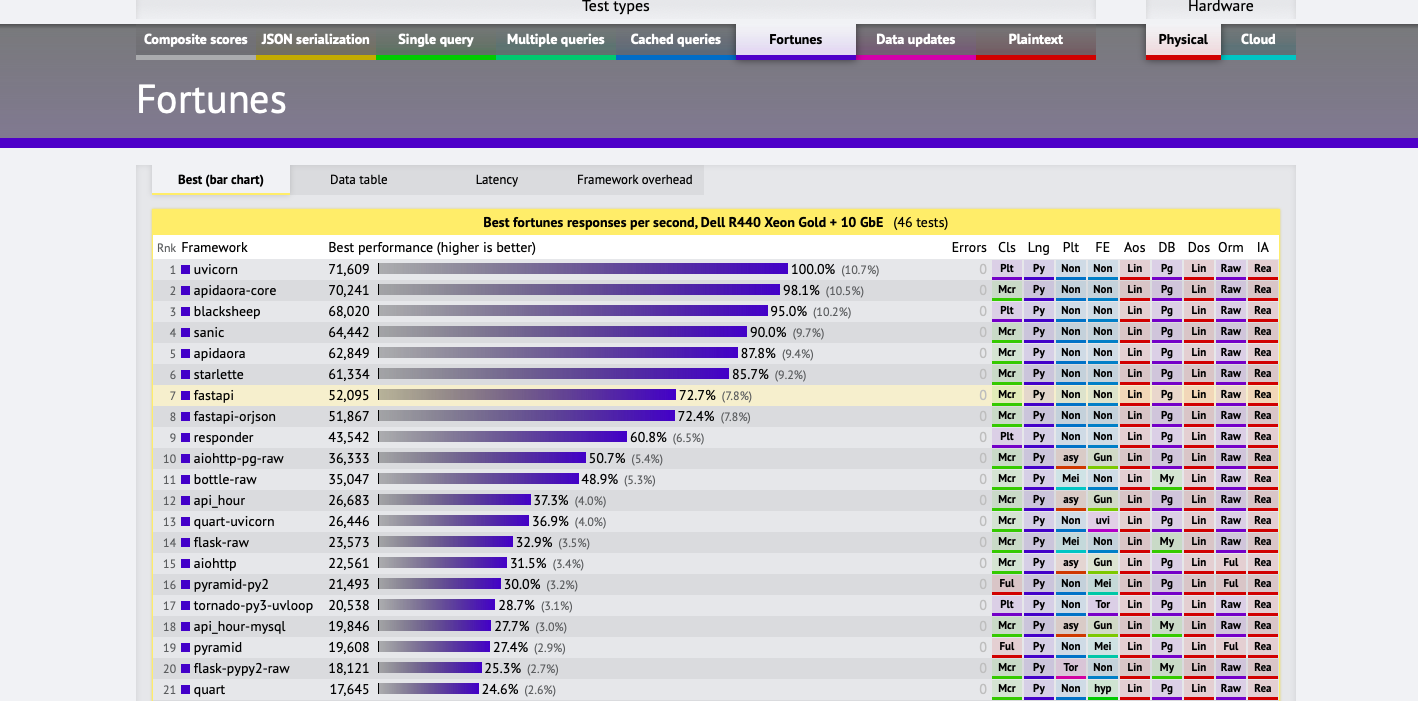
* **Performance**: Built over ASGI - (Asynchronous Server Gateway Interface) instead of WSGI - (Web Server Gateway Interface)
* **Fast to code:** Increase the speed to build the APIs
* **Documentation:**  Auto generates the documentation while developing the API

Fig1: Auto generated document by FastAPI

* **Data validation:** Built in data validation that can detect invalid datatype during the run and returns the reason for bad input in JSON format. Pydantic is used for data validation.
* **Based on open standards:** Uses OpenAPI for API creation, including declarations of path operations, parameters, body requests, security, etc.
* **Security:** Supports all the security standards defined in the OpenAPI standards, such as HTTP Basic, OAuth2, etc.
* **Support:** Has small but prompt community to support. Additionally the user documentation is very much detailed.
* **Benchmarks:** Comparing to other frameworks here is the performance benchmark result publishing in <https://www.techempower.com/benchmarks/>

Source: <https://www.techempower.com/benchmarks/>

Fig2: Benchmark results on best response per second

# Installation

pip install fastapi uvicorn

# or

poetry add fastapi uvicorn

pipenv install fastapi uvicorn

conda install fastapi uvicorn -c conda-forge

Note: FastAPI does not have a built-in development server, so an ASGI server like Uvicorn or Daphne is required. We will look further using uvicorn.

# Usage

**Hello World:**

Lets create basic hello world program, where the server will return string by calling the root endpoint.

Code: fastapi\_runner.py

import uvicorn

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

def home():

return {"Hello": "World"}

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run("fastapi\_runner:app")

Output:

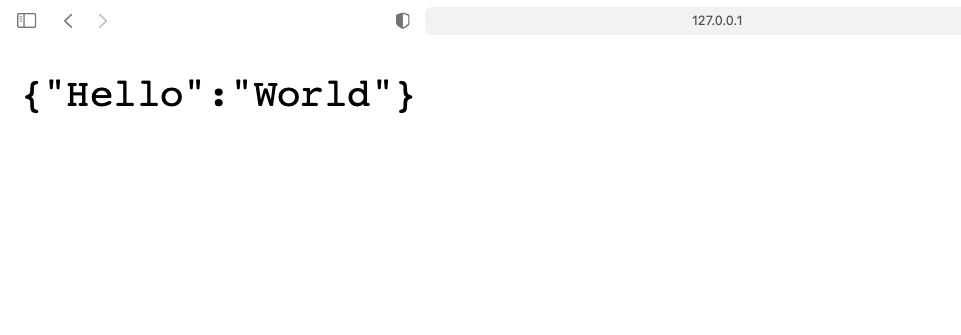
Url: http://127.0.0.1:8000/

Fig3: Response from root “/“ endpoint running using FastAPI

Url: http://127.0.0.1:8000/docs

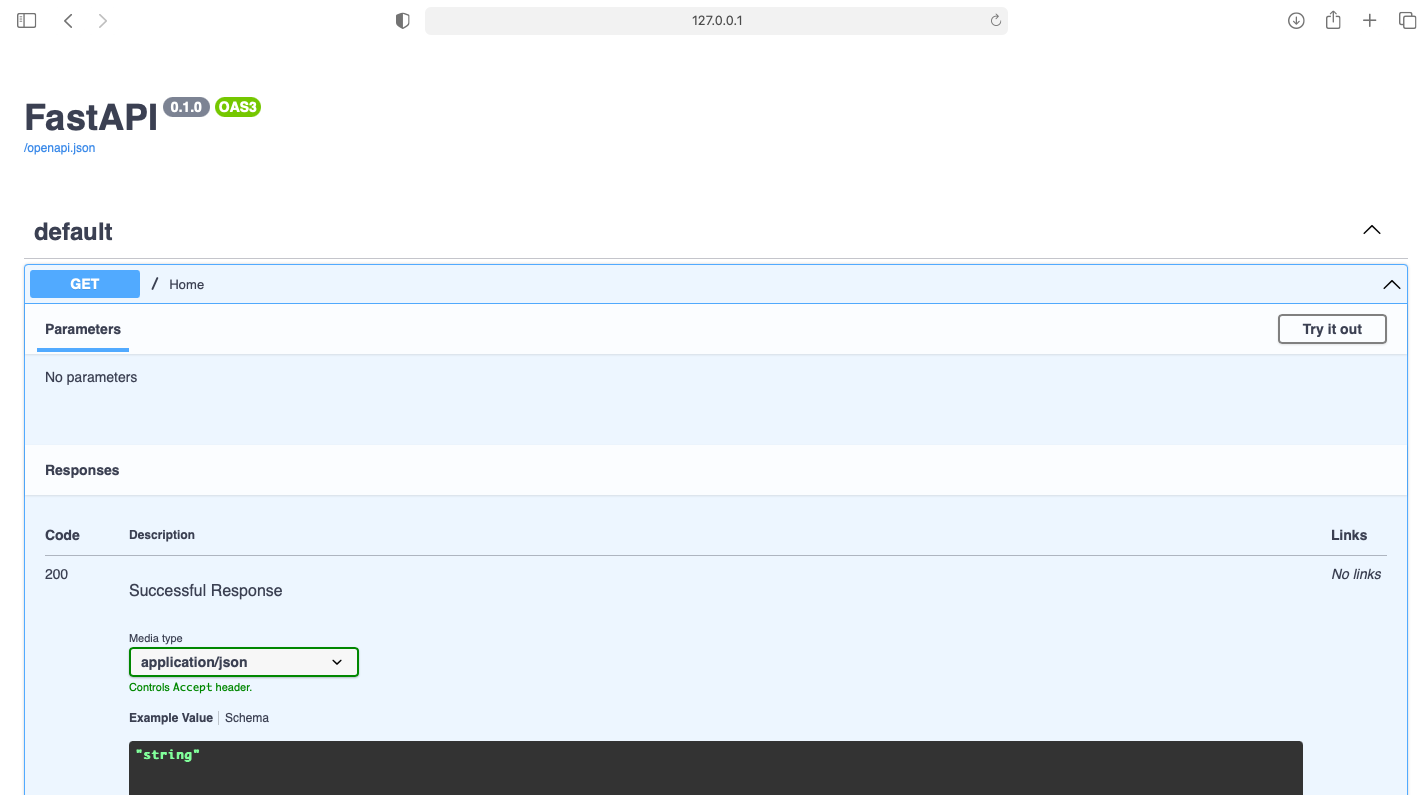


Fig4: FastAPI documentation autogenerated for the root endpoint replying the HelloWorld

**Create an item:**

Let's create post method, which expects gets body with list of attributes. Also we will see if pass invalid input the validation is internally handled by FastAPI.

Code:

import uvicorn

from fastapi import FastAPI

from pydantic import BaseModel

from typing import Optional, Set

app = FastAPI()

class Item(BaseModel):

name: str

description: Optional[str] = None

price: float

tax: Optional[float] = None

tags: Set[str] = []

@app.post("/items/", response\_model=Item, summary="Create an item")

async def create\_item(item: Item):

"""

Create an item with all the information:

- \*\*name\*\*: each item must have a name

- \*\*description\*\*: a long description

- \*\*price\*\*: required

- \*\*tax\*\*: if the item doesn't have tax, you can omit this

- \*\*tags\*\*: a set of unique tag strings for this item

\f

:param item: User input.

"""

return item

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run("fastapi\_runner:app")

Documentation:

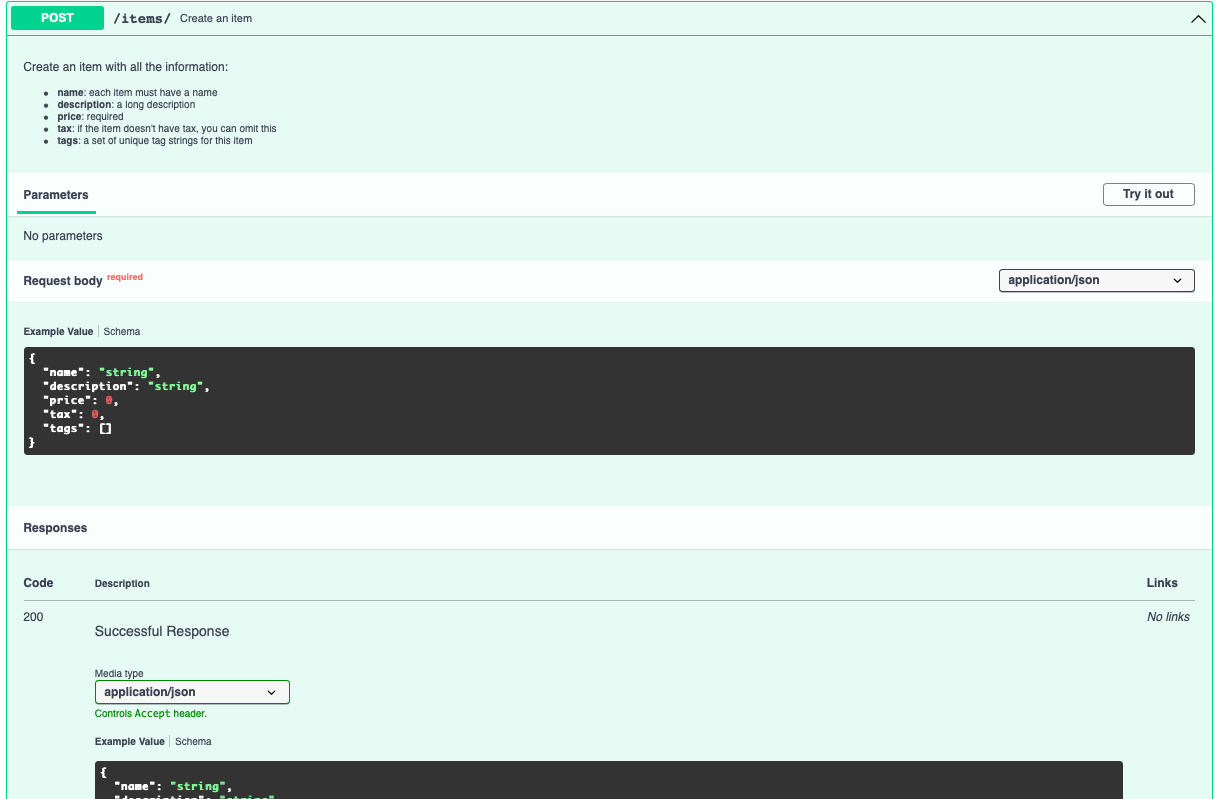
We could see the auto generator document from the summary and function comments.

Fig5: FastAPI POST documentation

Invalid Input:

Passing string instead of float in “price” variable.

{

"name": “first item",

"description": “first item description",

"price": "!2",

"tax": 123,

"tags": ["#fastapi", "#python"]

}

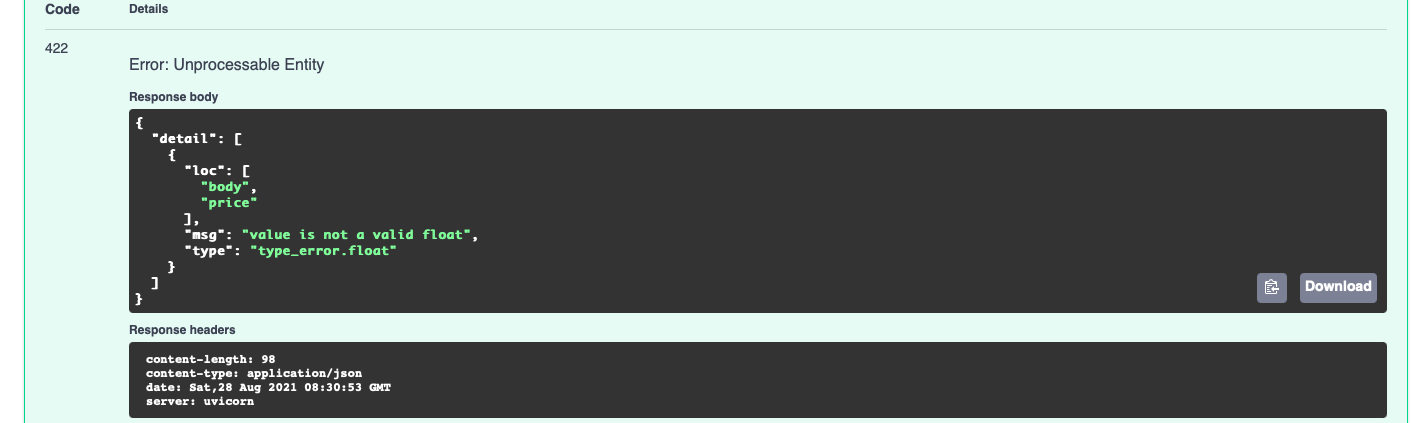
Error Response:

Fig6: FastAPI POST invalid error response

**Async Example:**

Below we will see the usage of query parameter and async & await functions in FastAPI.

Code:

import uvicorn

from fastapi import FastAPI

from pydantic import BaseModel

from typing import Optional, Set

import asyncio

app = FastAPI()

async def goto\_sleep(text, sleeptime):

print(f"before sleep {text}")

await asyncio.sleep(sleeptime)

print(f"after sleep {text}")

return f"{text} {sleeptime}"

@app.get("/slow")

async def slowresponse(st: int = 5):

print("below slowresponse starts")

sleptfor\_one, sleptfor\_two = await asyncio.gather(

goto\_sleep("one", st),

goto\_sleep("two", st),

)

print("after slowresponse {} {}".format(sleptfor\_one, sleptfor\_two))

return {"slow": "completed"}

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run("fastapi\_runner:app")

Input:

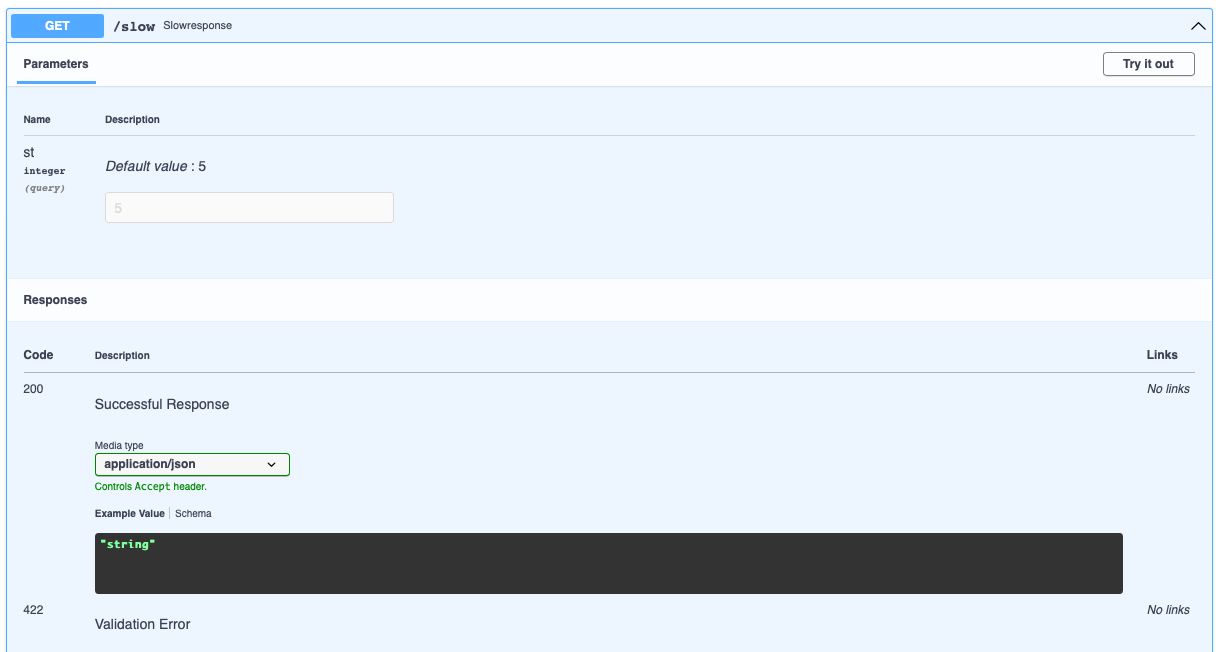
From the source code, we are expecting the value for query parameter “st” and also have default value set as 5 taken from the “slowresponse” function argument.

Fig7: FastAPI GET query documentation

Output:

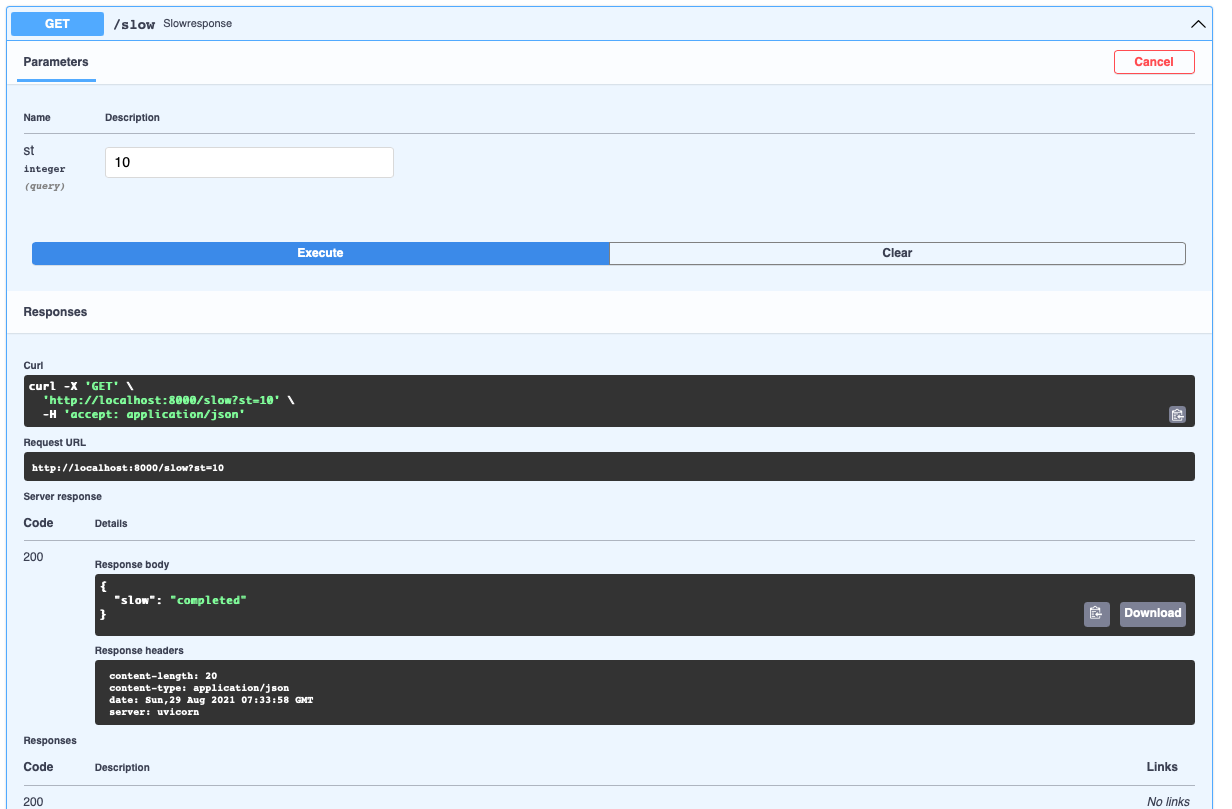
Here we could see the output of calling the “/slow” endpoint.

Fig8: FastAPI GET response documentation

Logs output:

Fig9: FastAPI async calls logs

# FastAPI vs Flask

| Features | FastAPI | Flask |
| --- | --- | --- |
| User Documentation | Natively yes | Not natively supported, need to use flask-swagger or such external module |
| SGI | Uses ASGI | Uses WSGI |
| Data validatiaon | Natively yes | Not natively supported |
| Beginner friendly | Yes | Yes |
| Concurrent programming | Natively yes using async and await | Not natively supported |
| Testing of APIs | Yes using fastapi.testclient.TestClient class | Yes using test\_client() function |
| HTTP Methods | Need to create separate decorator for each method  Eg: @app.get(“/“), @app.post(“/”) | Can combine multiple methods in single call.  Eg: @app.route(“/“, methods=[“GET”, “POST”] |
| Templates | Not natively supported, need to install Jinja.  But also having HTMLResponse to support html response | Natively yes, by importing render\_template |
| CORS | Natively yes | Not natively supported |
| Authentication | Natively yes, using fastapi.security | Not natively supported, server third party modules are available |

Table1: FastAPI vs Flask

# Cons

* Relatively new
* Small community compared to other frameworks
* Not a con but from the blogs mostly seems to be famous in ML community

# Conclusion

FastAPI is natively new, it is save lots of time in building the Web API easily and with user friendly documentation.

# Reference

* <https://fastapi.tiangolo.com>
* <https://testdriven.io/blog/moving-from-flask-to-fastapi/>
* <https://www.pluralsight.com/tech-blog/porting-flask-to-fastapi-for-ml-model-serving/>
* <https://analyticsindiamag.com/fastapi-vs-flask-comparison-guide-for-data-science-enthusiasts/>a