**REST API Calculator**

**Dhruv Krishnamachari**

**AIM:** To build a RESTful API using Python and implement simple calculation functions using Path and Query Parameters.

**Tools Used:**

* **Editor:** Visual Studio Code
* **Python:** v3.8.6
* **Postman:** To test API routes

**Libraries:**

* **FastAPI:** A relatively new, robust and versatile library for creating APIs
* **Uvicorn:** A Python based web server used to run our API
* **Math:** Python’s in-built math class from the standard library

**Research:**

The term “API” is an acronym for Application Programming Interface. APIs are intermediary or middle man software that allow applications to talk to each other. The most common use for APIs is to request data from a server and return it in a raw format. This can be a JSON object, raw text, markdown or more. REST stands for Representational State Transfer. This term was coined by Dr. Roy Fielding in his PhD dissertation back in 2000. REST is not a specific software, tool or technique. Instead, it is a set of guidelines defined by him for how to design the architecture of software like APIs for maximum efficiency and user friendliness. Since then, REST has become the go-to constraint format when it comes to designing APIs.

Python has a few libraries which can be used to design APIs. In fact, it can even be done with the in-built “requests” library. However, I have opted to use an external library to save time instead of re-inventing the loop. In the last few years, an open-source library by GitHub user Tiangolo has been making waves in the industry. This is the FastAPI library that I have chosen to use. I had been meaning to use it for some time and this project presented the perfect opportunity to do so. FastAPI isn’t just fast as the name implies, but provides some great features under the hood. It automatically documents new API endpoints that you write and even provides its own testing environment where you can interact with and test the endpoint, powered by Swagger UI. Visiting the root directory/docs opens a list of endpoints written in the API, and expanding any one of them reveals a testing field, schemas, possible HTTP codes and so on.

Unlike Flask, a close competitor to FastAPI, it does not come with its own in-built proprietary web server. This is not entirely a bad thing as it lessens the size and load and increases the efficiency. However, we do need a webserver to run the API in so we can interact with the endpoints and test them. For this purpose, I have chosen Uvicorn. There are a few other options like Gunicorn or Hypercorn, however Uvicorn is ASGI (Asynchronous Standard Gateway Interface) based. Gunicorn is WSGI (Web Server Gateway Interface) based, which is the predecessor to ASGI. Hypercorn is also ASGI based, but more complex than we need.

Postman needs no introduction, it is a tried, tested and true tool to test and interact with APIs built in any language. I used this tool to test every single endpoint in the calculator as it provides more functionality than FastAPI’s autodocs. The Math library was used for certain complex calculation endpoints. Certain functions like sin, cos and tan return values that may seem incorrect but are a result of computer limitations. For example, cos 90 (cos pi/2) would return 6e-17 instead of 0 due to floating point errors, but by rounding it can be equated to 0. This problem applies to other trigonometric functions, but they cannot all be simply rounded.

**Summary of the attached files:**

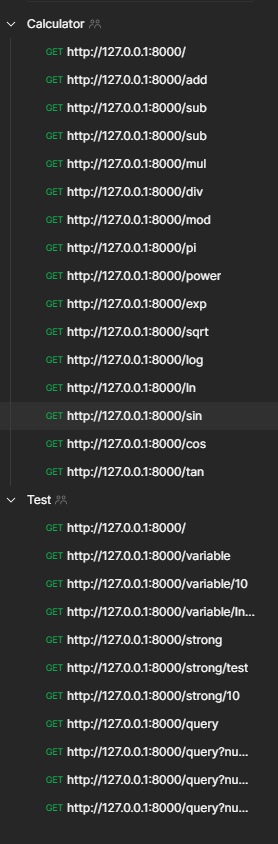
I have attached a requirements.txt file which can be run with the command “pip install requirements.txt” to install all the requirements to run the code attached. I have created a file called “demo.py” in which I have my earliest testing code and explanation about Path and Query parameters. The other file, “app.py” is where the main endpoints for the calculator reside. To run either file, use the following commands:

* demo.py: “uvicorn demo:app --reload”
* app.py: “uvicorn app:app --reload”

The pattern is “uvicorn filename:instance\_name –reload. The reload flag reloads the webserver every time any change is made to the file which is quite convenient.

Note: FastAPI comes with a proprietary HTTPException class to catch and raise custom errors instead of the in-built error schema. However, in this case I need to catch whether the function’s arguments are present or not. I initially implemented code to catch and raise this with a code 422 (as that is what FastAPI itself raises), however, the internal error schema always seems to trigger before the exception itself. As such, I have not handled errors for mostly anything; the errors can be read from the returned JSON regardless.

**Postman screenshot for all the endpoints in a list:**

****

**Postman screenshots for each endpoint:**

I will include the screenshots for each individual endpoint in relevant folders within the project folder to prevent cluttering the word doc.

**References:**

* Google
* FastAPI examples/home
* FastAPI error handling docs
* Uvicorn docs
* StackOverflow
* Python’s Math docs
* YouTube for the history of RESTful APIs