Flask based API development

Python is a programming language that allows you to create a wide range of web apps, websites, APIs, and desktop apps using frameworks. But exactly, what is a framework? A framework is a set of libraries and modules that help developers in the creation of complex, scalable, maintainable, and reliable applications. A framework typically assists in the creation of reusable code and extensions. Flask, Tornado, Pyramid, and Django are some Python frameworks. We'll look at where Flask shines and how to use it to develop a simple Python API in Python.

Introduction to Flask

Flask is basically a micro web application framework written in Python. Flask is commonly used by developers to create web applications, manage HTTP requests, and render templates. By “micro web application” we mean that it is not a full-stack framework.

The term "micro" refers to a core component that is both simple and extensible. Flask is easy to work with on any project because it is lightweight — it's easier to find and fix errors than a full-stack framework. Flask can also be used to create stunning products as well as impressive applications. Flask is used by several large firms such as:

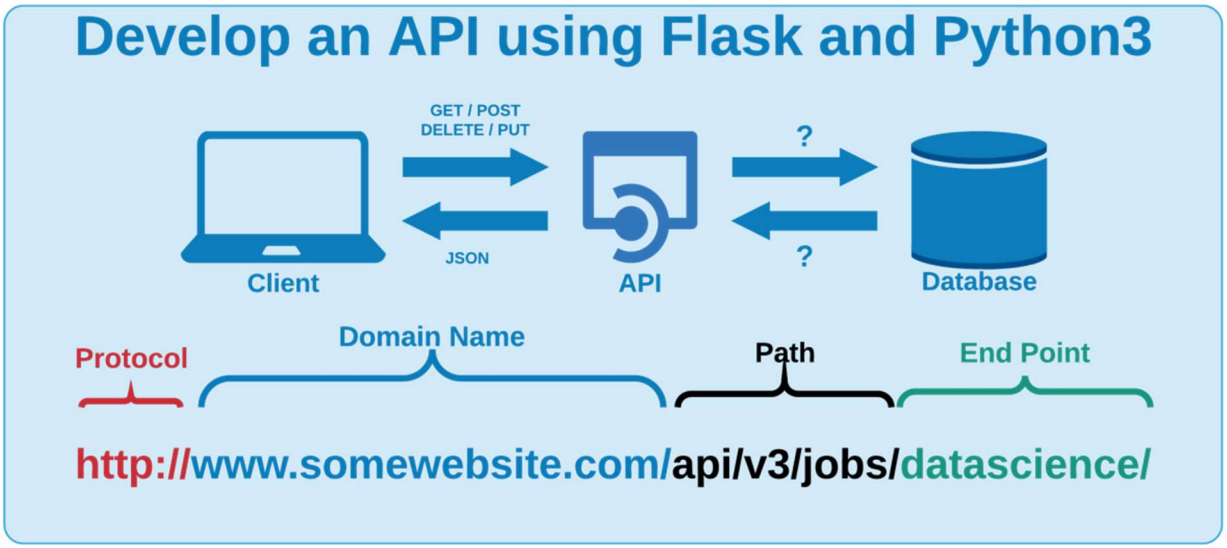
Coming to the history of Flask, it was created as an April Fool's joke in 2010 by Armin Ronacher, the leader of the Python organization Pocco. Flask earned a lot of attention in a 2018 Python Developers survey, despite its lack of popularity at the time of development. It had gained a lot of traction on GitHub by January 2020. Flask is built on the Pocco projects Werkzeug, WSGI toolkit, and Jinja2 engine.

- Werkzeug: One of the foundations of the Flask framework for implementing requests, response objects, and other utility operations.

- WSGI toolkit: WSGI stands for Web Server Gateway Interface. WSGI is a specification that specifies how web servers communicate with Python web applications or frameworks.
- Jinja2 engine: Jinja2 is a modern-day templating language for Python developers. That can be used with data sources to create dynamic web pages.

Why API development with Flask?

API stands for Application Program Interface. In simple words, it's just a small piece of code that's available in such a way that any application written in any language, including Java, Python, and Node.js can send a request to retrieve some output for some certain input. It just acts as an interface between the underlying application and the rest of the world.



Why do we need Flask if we already have the Django framework?

Flask is well-known for creating web apps and APIs due to its well-defined project structure and built-in tools? This is because it's like an empty canvas for creating Python-based apps, with no project layout and few dependencies. Flask can make suggestions for libraries and tools that can be used for development.

Pros

- Almost all the parts of flask are open to change and we can alter it according to our requirements in a variety of ways.

- Using Flask for web development allows for unit testing through its integrated support, built-in development server, fast debugger, and restful request dispatching.
- Being slightly lower level with fewer levels of abstraction, it allows you to have a granular control, hence leading to superior performance.
- It provides the ability to create multiple Flask applications or servers, distributed across a large network of servers, each with specific purposes, creating more efficiency and testability.

Cons

- Its feasible for small scale applications but isn't suitable for large scale applications.
- On the flip side of being simpler, it's not very opinionated, hence its not very standardized.
- You don't have a full toolset underneath you. So you may need to build more on your own or search out extensions/libraries from the community.
- As compare to flask faster frameworks are available which are equivalent in to flask such as FastAPI

Where to start?

Prerequisite: Python, Virtulenv

Why do we need a virtual environment?

A virtual environment is used to create an isolated Python environment for different projects. We create virtual environments because different projects have different dependencies. Also, it helps to keep the global packages folder clean.

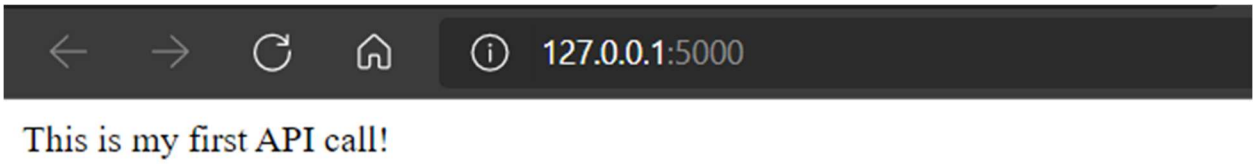
```python
from flask import Flask
app = Flask(__name__)


@app.route('/')
def index():
    return 'This is my first API call!'

if __name__ == '__main__':
   app.run(debug = True)
```

Code Explanation: First, we are importing the flask module into our application and then defining the API route. The @ is used for defining the API route. We're passing /, which means this is our base route.

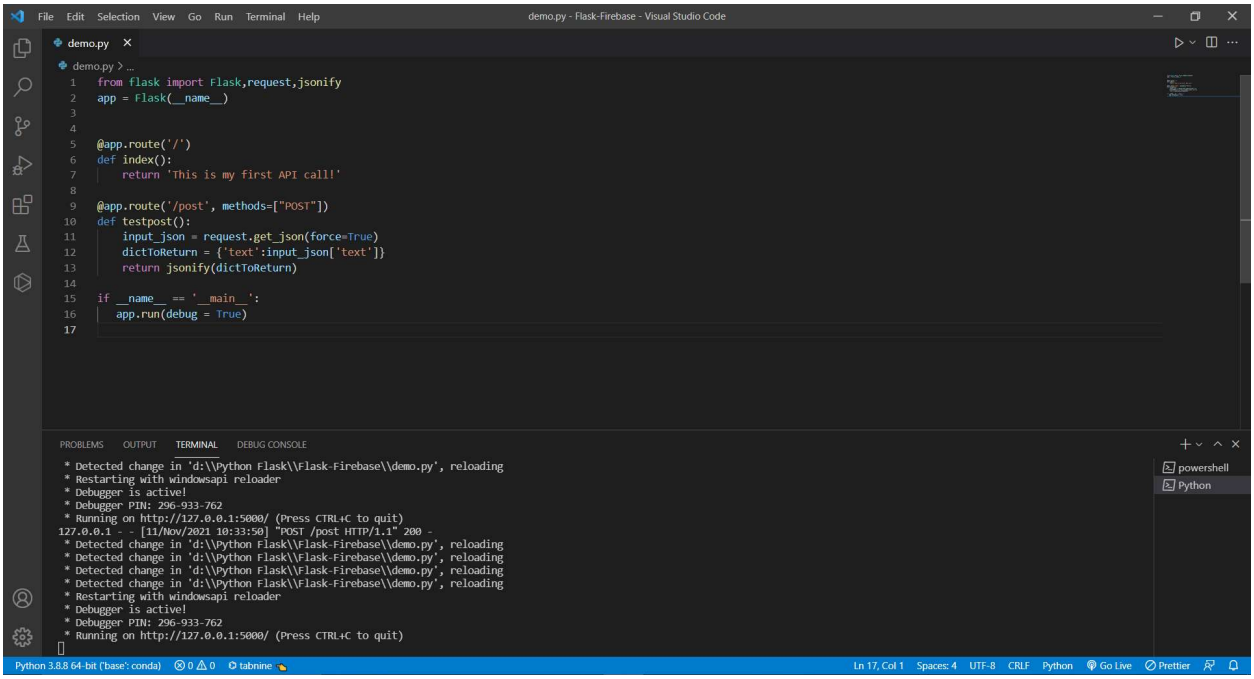
This is sample program that gives following output.



POST APIs

Flask makes it very easy to program different commands for various HTTP methods like POST, GET, PUT, etc. In the above code, you can see there's the function route. The second parameter passed to this function is actually the method of the route. If nothing is passed then, it is GET.

But we also have to import two additional modules named request and jsonify used to fetch the params and JSON conversion.



Code Explanation: Here, we have imported some more modules from Flask, like request and jsonify. We are using request to get the data which the user is sending,

and we're using jsonify for converting dictionaries to JSON. We have added one more route that is /post and also passing POST as a list and returning back what the user is sending in the parameters.

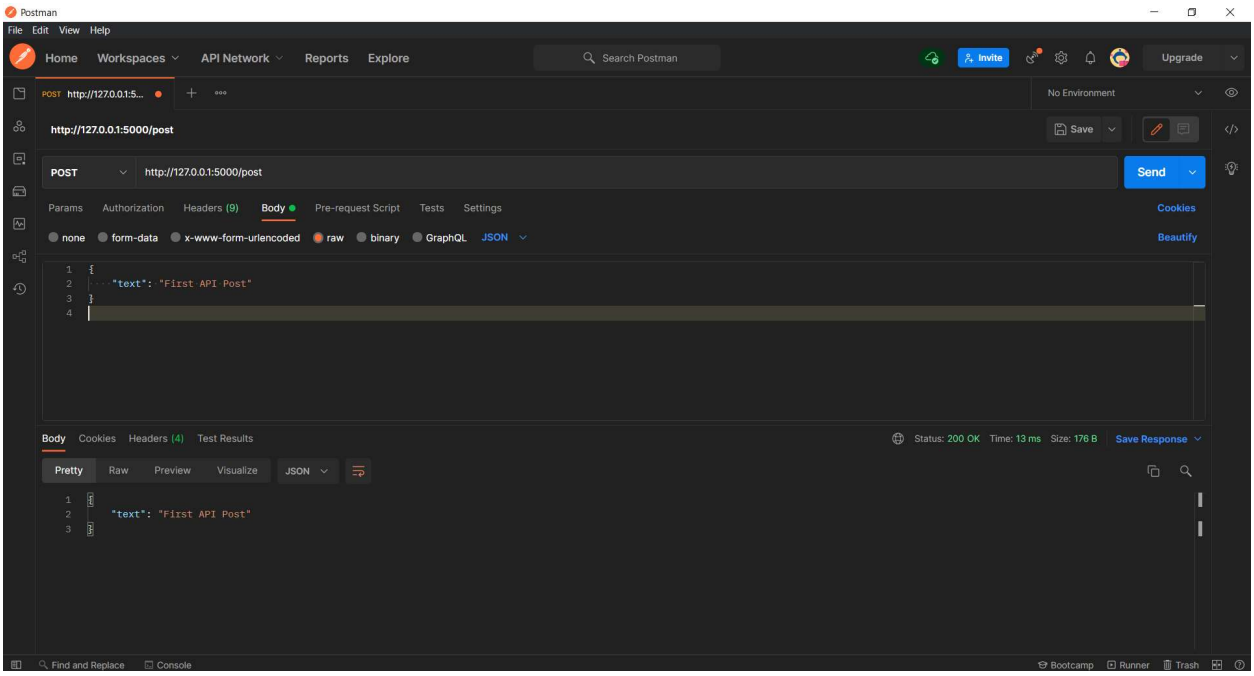
To test this API we are using postman.

Now head over to the API testing tool and hit URL:

http://127.0.0.1:5000/post with parameters:

```
{
   "text": "First API Post"
}
```

You should see the below response:



Conclusion

Now we're done with the most basic Flask tutorial with one GET and one POST API endpoint. This example was just a gist so that you can understand the basics of Flask.

Flask is a very powerful tool, and on an advanced level, you can achieve a lot of things with it. For example, you could add authentication with JWT or OAuth. You can also connect the code with a MySQL backend and perform CRUD operations.

I hope you learned something new from this article, and hopefully, I made it easier for you to understand Flask and API creation basics.