

Forum:[Hack Forums](#)

Thread: [Python API Web framework](#)

amethyst - May 16, 2020, 8:45 pm

which is the best python web framework to build APIs?

4Txe4y33C9zvNhMj - May 17, 2020, 10:30 am

Probably Python Flask, they have an extension for RESTful APIs

amethyst - May 17, 2020, 10:59 am

(May 17, 2020 10:30 AM)4Txe4y33C9zvNhMj Wrote: [\[CITE\]](#) Probably Python Flask, they have an extension for RESTful APIs

I see Flask sounds good

Lindros - May 17, 2020, 10:55 pm

I like Django as a web framework in Python but that's cause i'm biased as it has been the only web framework i've been exposed to so far in Python. It seems really good though, some people complain that it is a little heavy but it comes out of the box with nice security features.

Athr - May 17, 2020, 10:59 pm

I've created some basic APIs using Flask. Its very handy as you can just return the data using the jsonify() function available in flask.

Example:

```
Code @app.route('/api', method = ['GET'])
def api():
    data = {'status': 1, 'data' : [0, 1, 2, 3]}
    return flask.jsonify(data)
```

amethyst - May 18, 2020, 2:08 am

(May 17, 2020 10:55 PM)Lindros Wrote: [\[CITE\]](#) I like Django as a web framework in Python but that's cause i'm biased as it has been the only web framework i've been exposed to so far in Python. It seems really good though, some people complain that it is a little heavy but it comes out of the box with nice security features.

(May 17, 2020 10:59 PM)Athr Wrote: [\[CITE\]](#) I've created some basic APIs using Flask. Its very handy as you can just return the data using the jsonify() function available in flask.

Example:

```
Code @app.route('/api', method = ['GET'])
def api():
    data = {'status': 1, 'data' : [0, 1, 2, 3]}
    return flask.jsonify(data)
```

I like Django I think it's an awesome framework many top-tier company use it for a lot of things and solid and stable but it can seem very complicated for beginners but it's good

Flask seems much more like a simple framework reminds me of node.js/express minimal framework

SolarPlexus - May 26, 2020, 12:37 am

I'm using django too. It's not heavy as people told.

synchronizing - May 26, 2020, 8:21 am

It depends on your use case. You have to ask yourself a few basic questions first:
Is the server going to process anything before replying?

How many connections can the server potentially have simultaneously?

What is the server serving?

Must the server respond with proper HTTP responses?

When programming an API endpoint you want to ensure that it is going to work for your use case. A few of the libraries mentioned here work on the bases of synchronous code (as opposed to asynchronous or parallel), and this has large effects on the way your server runs.

Frameworks like Flask and Django do not provide methods to compute or process other portions of code without blocking the thread. In other words your server is incapable of receiving/processing/replying simultaneously. That means if user A requests a resource from the server and user B requests something 1ms afterwards, user B must wait until the entire chain (receiving/processing/replying) is done for user A before the server even bothers with user B. This is something that doesn't really matter when it comes to a small API endpoint that only you will be using, but it does matter when you are exposing this endpoint to the world. Imagine if processing takes 2s, and 10 users are in line... the last user must wait 20 seconds before receiving a reply.

You can use uwsgi with both Django and Flask (out of the box) to minimize this issue by spawning multiple Flask/Django processes at the same time. This might allow 5-15 parallel requests to your application, which is a large improvement, but not optimal if for large scale. If you are serving a lot of static information (no process needed), look into a load balancer/reverse proxy like nginx to cache static content. If you intend on making a server that reaches out to external resources (say an unofficial Twitter RESTful API that scrapes/process/replies users information), use an asynchronous framework like Sanic... and so on.

Point is: writing an API is fairly easy, but scaling is fairly hard. Your use case is what's important, and you have to write code around that.

vorticalbox - May 27, 2020, 3:20 am

I personally like <https://sanic.readthedocs.io/en/latest/>

4Txe4y33C9zvNhMj - May 28, 2020, 6:21 am

(May 26, 2020 08:21 AM)synchronizing Wrote: [\[CITE\]](#) It depends on your use case. You have to ask yourself a few basic questions first:

Is the server going to process anything before replying?

How many connections can the server potentially have simultaneously?

What is the server serving?

Must the server respond with proper HTTP responses?

When programming an API endpoint you want to ensure that it is going to work for your use case. A few of the libraries mentioned here work on the bases of synchronous code (as opposed to asynchronous or parallel), and this has large effects on the way your server runs.

Frameworks like Flask and Django do not provide methods to compute or process other portions of code without blocking the thread. In other words your server is incapable of receiving/processing/replying simultaneously. That means if user A requests a resource from the server and user B requests something 1ms afterwards, user B must wait until the entire chain (receiving/processing/replying) is done for user A before the server even bothers with user B. This is something that doesn't really matter when it comes to a small API endpoint that only you will be using, but it does matter when you are exposing this endpoint to the world. Imagine if processing takes 2s, and 10 users are in line... the last user must wait 20 seconds before receiving a reply.

You can use uwsgi with both Django and Flask (out of the box) to minimize this issue by spawning multiple Flask/Django processes at the same time. This might allow 5-15 parallel requests to your application, which is a large improvement, but not optimal if for large scale. If you are serving a lot of static information (no process needed), look into a load balancer/reverse proxy like nginx to cache static content. If you intend on making a server that reaches out to external resources (say an unofficial Twitter RESTful API that scrapes/process/replies users information), use an asynchronous framework like Sanic... and so on.

Point is: writing an API is fairly easy, but scaling is fairly hard. Your use case is what's important, and you have to write code around that.

That's not true. No clue about Django but Flask has been multithreaded for years. It can definitely handle more users.

I doubt he's going to scale it much anyway.

synchronizing - May 28, 2020, 9:33 am

(May 28, 2020 06:21 AM)4Txe4y33C9zvNhMj Wrote: [\[CITE\]](#) That's not true. No clue about Django but Flask has been multithreaded for years. It can definitely handle more users.

I doubt he's going to scale it much anyway.

Just did some research, and you are right. I haven't used Flask in years now, and back then it was not multithreaded. Seems since then the underlying Werkzeug server has been updated and now defaults to multithreading for individual requests, which is super nice to know. Will edit post I can't edit the post, but thank you for heads up!