Python’s Web Frameworks: A Comparison

Shayan Shojaei

Spring 2024

Contents

1 Introduction

This report presents a comparative analysis of three prominent web frameworks –Django, Flask, and FastAPI—by implementing a basic CRUD (Create, Read, Update, Delete) application and evaluating their performance across two different database systems. The study aims to provide insights into the efficiency, scalability, and ease of use of each framework. Django, known for its comprehensive features and built-in administrative interface, is contrasted with Flask's lightweight and modular design, and FastAPI's modern, asynchronous capabilities. Performance metrics such as response time, throughput, and resource utilization are analyzed under varying workloads to determine the frameworks' suitability for different application scenarios. This evaluation offers valuable guidance for developers in selecting the appropriate framework based on specific project requirements and performance criteria.

2 Overview

In this study, APIs were implemented and deployed using Django, Flask, and FastAPI, adhering strictly to the guidelines provided in each framework's official documentation. No additional optimizations were applied, ensuring a straightforward and minimalistic approach for

Django

For Django, a new project was created using the `django-admin startproject` command, followed by the creation of two apps for MongoDB and PostgreSQL respectively, with `python manage.py startapp`. Models were defined in the `models.py` file and URL routing was configured in `urls.py`, and the built-in development server was used for deployment.

Flask

In Flask, the application was structured by creating a module for Postgresql and MongoDB each, with routes defined using the `@app.get`, etc. decorator. SQLAlchemy was used for postgres interactions, motor for MongoDB, and the CRUD operations were implemented directly within the route functions. The application was run using Flask's built-in server by executing the script with `flask run main.py` which is a WSGI server and not a ASGI server, resulting in lower performance than a ASGI server.

FastAPI

FastAPI's implementation involved defining route functions with Python's async capabilities and using Pydantic models for data validation. The fastapi package was employed to create the app, and CRUD routes were defined using the `@app.get`, `@app.post`, `@app.put`, and `@app.delete` decorators. The application was deployed using FastApi's own implementation of an ASGI server, by running `fastapi run app.py`.

3 Benchmarks

The benchmarks for evaluating Django, Flask, and FastAPI were set up using Grafana's K6, a modern load testing tool. Five distinct types of tests were designed to comprehensively assess the performance of each framework. The "create" test involved generating as many items as possible using the POST method within a 30-second window, focusing on the frameworks' ability to handle high-volume data insertion. The "read" test measured the efficiency of each framework in retrieving a single record repeatedly over 30 seconds. Similarly, the "update" test evaluated the frameworks' performance in modifying a record multiple times within the same time constraint. The "crud" test was more complex, involving the creation, reading, updating, and deletion of a record in succession, aiming to simulate a typical workload scenario within the 30-second limit. Lastly, the "crud ramp up" test incrementally increased the number of virtual users from 20 to 100 over 30 seconds, then maintained this load for an additional 1 minute and 30 seconds, followed by a 20-second ramp-down period. This test was designed to assess how each framework handles increasing and sustained loads, providing insights into scalability and robustness under stress.

4 Reports

4.1 Create

```
POST /<database>/
```

Database	Framework	# VUs	# Iterations	RPS	Failure %
MongoDB	Django	10	266	8.692	0
		100	1606	49.741	0
	Flask	10	11169	372.000	0
		100	10823	357.103	0
	FastAPI	10	39130	1304.079	0
		100	83077	2768.474	0
Postgresql	Django	10	12711	423.400	0
		100	10468	345.937	0
	Flask	10	16217	495.700	0
		100	16440	448.550	0.004
	FastAPI	10	3309	110.152	0
		100	13306	442.494	0

4.2 Read

```
GET /<database>/<id>
```

Database	Framework	# VUs	# Iterations	RPS	Failure %
MongoDB	Django	10	590	19.415	0
		100	3842	122.790	0
	Flask	10	16272	484.076	0.0005
		100	16352	411.493	0.005
	FastAPI	10	113067	3768.711	0
		100	121802	4057.567	0
Postgresql	Django	10	12783	425.469	0
		100	10896	361.387	0
	Flask	10	12442	319.839	0
		100	16443	418.668	0.004
	FastAPI	10	13147	437.808	0
		100	25608	852.717	0

4.3 Update

```
    PUT /<database>/<id>
```

Database	Framework	# VUs	# Iterations	RPS	Failure %
MongoDB	Django	10	336	10.843	0
		100	2353	73.926	0
	Flask	10	10290	342.723	0
		100	10317	340.439	0
	FastAPI	10	33662	1121.816	0
		100	105412	3503.162	0
Postgresql	Django	10	10753	358.242	0
		100	9676	321.159	0
	Flask	10	16247	450.784	0
		100	16362	408.542	0.00006
	FastAPI	10	3509	116.634	0
		100	8052	267.447	0

4.4 CRUD

```
    POST /<database>/
    GET  /<database>/<id>
    PUT  /<database>/<id>
    DEL  /<database>/<id>
```

Database	Framework	# VUs	# Iterations	RPS	Failure %
MongoDB	Django	10	86	10.248	0
		100	600	68.080	0
	Flask	10	3247	478.994	0
		100	3303	408.065	0
	FastAPI	10	12976	2161.176	0
		100	24026	3992.831	0
Postgresql	Django	10	3116	414.551	0
		100	2499	327.052	0
	Flask	10	3252	444.073	0.0004
		100	3301	406.211	0.0038
	FastAPI	10	1240	205.847	0
		100	3573	589.888	0

4.5 CRUD – Ramp Up

Same as CRUD, but with a ramp-up of 20 to 100 VUs over 30 seconds, then maintaining 100 VUs for 1 minute and 30 seconds, followed by a 20-second ramp-down.

Database	Framework	# Iterations	RPS	Failure %
MongoDB	Django	1366	38.018	0
	Flask	9813	330.328	0
	FastAPI	98488	3517.344	0
Postgresql	Django	10571	301.974	0
	Flask	13070	405.767	0.0019
	FastAPI	20884	745.829	0

5 Analysis

The analysis of the performance benchmarking reports reveals several important findings. The combination of MongoDB with Django emerged as the worst performer, while the pairing of MongoDB with FastAPI proved to be the best. Despite these results, PostgreSQL should not be discounted, as its capabilities extend beyond mere read and write performance, offering valuable features that may be crucial depending on the application's requirements.

Flask encountered unexpected bottlenecks when interfacing with PostgreSQL, suggesting potential areas for performance improvement. However, the benchmarking aimed to evaluate the frameworks in their default configurations without optimizations. Consequently, this investigation did not delve into resolving these issues. Notably, Flask was the only framework to experience failed requests. Although the number of failures was minimal, the absence of such failures in other frameworks casts doubt on Flask's reliability in comparison.

A gradual increase in the number of virtual users (VUs) slightly enhanced the performance of PostgreSQL when combined with FastAPI and Django, which is consistent with the behavior of PostgreSQL's connection pooling. Conversely, Django did not perform well with MongoDB. This may be attributed to the use of a third-party library for MongoDB integration, as well as Django's design philosophy of abstracting database interactions, which limits developer control over connection management.

The performance disparity between Django and Flask, both using WSGI servers, and FastAPI, which uses an ASGI server out-of-the-box, is notable. While it is possible to deploy Django and Flask with an ASGI server, doing so would require additional setup and middleware, which was not within the scope of these benchmarks. This intrinsic difference in server architecture likely contributes to FastAPI's superior performance in the tests conducted.

6 Conclusion

In conclusion, the benchmarking results indicate that FastAPI with MongoDB provides the best performance among the tested combinations, while Django with MongoDB performs the worst. Although PostgreSQL's performance was not the highest, its robust feature set makes it a viable option for applications that require advanced database functionalities. The unexpected bottlenecks encountered by Flask with PostgreSQL suggest room for optimization, but Flask's reliability issues, as evidenced by failed requests, undermine its competitiveness. The inherent performance advantage of FastAPI, due to its ASGI server architecture, highlights the impact of server choice on framework efficiency. These findings underscore the importance of considering both performance metrics and specific application needs when selecting a web framework and database combination.