WROCŁAW UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF ELECTRONICS

FIELD: Informatyka Techniczna (INF) //TODO
SPECIALIZATION: Internet Engineering (INE)

MASTER OF SCIENCE THESIS

Comparison analysis and efficiency evaluation of back-end frameworks in database applications

Analiza porównawcza i ocena wydajności frameworków back-endowych w aplikacjach bazodanowych

AUTHOR:
Marcin Wojciechowski

SUPERVISOR:
Dr inż. Paweł Głuchowski W4/K9

GRADE:

WROCŁAW 2021

Contents

1 Introduction **3**

2 Technology description **4**
2.1 Django . . . 4
2.1.1 Overview . . . 4
2.1.2 Architecture . . . 4
2.1.3 Requirements . . . 5
2.2 ExpressJS . . . 5
2.2.1 Overview . . . 5
2.2.2 Architecture . . . 5
2.2.3 Requirements . . . 5
2.3 ASP.NET . . . 5
2.3.1 Overview . . . 5
2.3.2 Architecture . . . 5
2.3.3 Requirements . . . 5
2.4 K6 and related tools . . . 5
2.5 Docker and Docker Compose . . . 6
2.6 PostgreSQL . . . 6

3 Criteria description **7**
3.1 Performance benchmark . . . 7
3.1.1 Scenarios . . . 7
3.1.2 Database snapshots . . . 7
3.1.3 Application isolation . . . 7
3.1.4 Test progress . . . 7
3.1.5 Software versions and hardware . . . 8
3.2 Security . . . 8

4 System design **9**
4.1 Database . . . 9
4.2 Application . . . 9
4.3 Environment . . . 9

5 Application implementation **10**
5.1 Django . . . 10
5.2 Express . . . 10
5.3 ASP.NET . . . 10

6 Results **11**
6.1 Performance 11
6.2 Security 11

7 Conclusion **12**

Bibliography **12**

Chapter 1

Introduction

Complex web applications keep becoming more popular in recent years. They are very easily accessible from any place in the world and can run on almost any modern device, as it requires only web browser and the Internet connection to run.

Web frameworks simplify the development process of web applications significantly improving developers productivity. There is a huge variaty of choices between the mentioned software, so choosing one may be a difficult task.

The choice of server-side framework is crucial, as it is responsible for handling sensitive data and plays a key role in the overall performance of the application.

The goal of this document is to focus on three of the most popular server-side frameworks and compare their performance under high load as well as basic security measurements. The results of this thesis will be important for web developers and architects, that need to decide on which framework should they choose for their application.

Web frameworks for this comparison were chosen from the list of Stack Overflow Developer Survey 2020 Web Frameworks popularity [1]. The chosen frameworks (and their respective languages) are:

- Express.js (JavaScript)
- ASP.NET (C#)
- Django (Python)

Chapter 2 describes chosen frameworks, including their architecture and requirements. For the definition of the criteria against which the results will be measured, see Chapter 3. Chapter 4 presents design of the system - database models, application structure and environment preparation. Chapter 5 shows framework specific application implementation details. Results of the tests and comparison of the technologies can be found in chapter 6. For final conclusion of this document, see chapter 7.

Chapter 2

Technology description

2.1 Django

2.1.1 Overview

Django lets you build deep, dynamic, interesting sites in an extremely short time. Django is designed to let you focus on the fun, interesting parts of your job while easing the pain of the repetitive bits [2]. Additionally the framework is being supported by a wide variety of libraries and frameworks, like Django Rest Framework, Django Celery, Crispy Forms.

2.1.2 Architecture

Django is based on MVC architecture:

- Model - a data structure, represented by a database
- View - responses visible in the browser
- Controller - connects Model and View together - describes how the data should be presented to the user

In Django application there are multiple files and at first it may not be obvious what their role is. Basic structure looks like this:

- apps.py - common to all django apps configuration file
- models.py - custom models
- serializers.py - define how our model objects should be converted into response
- views.py - custom controllers, which is unintuitive for most of people; as the developers explain, in their interpretation of MVC the view describes which data gets presented to the user [3]
- urls.py - defines which endpoint responds to given controller

Templates, which are not mentioned above, are the Django's custom views - in our case, I am going to be using build in json parsers.

2.1.3 Requirements

To install and run a simple Django project, two main things are required:

- pip (easiest install method)
- Python

2.2 ExpressJS

2.2.1 Overview

2.2.2 Architecture

2.2.3 Requirements

2.3 ASP.NET

2.3.1 Overview

2.3.2 Architecture

2.3.3 Requirements

2.4 K6 and related tools

For testing the performance of applications, I chose a tool named k6. It is a modern load testing tool written in Golang, which provides clean and well documented APIs for writing and running tests, while still being easily configurable to the developers needs. Test logic and configuration options are both to be written in JavaScript, which allows developers for using JavaScript modules, which aids in code reusability. The creators of k6 prepared two types of execution:

- local, through command line interface
- and cloud, which is a commercial SaaS product, made to make performance testing in bigger applications easier.

For the sake of this experiment, local testing has fulfilled all expectations.

Installation on Ubuntu operating system is fairly simple and all necessary commands were described in the documentation. However, to make the testing simpler, a k6 Docker image was used, that together with Docker Compose allowed to create a single script that would handle all test cases as described in the following section.

K6 allows to create visualizations, using built-in InfluxDB and Grafana integration, where InfluxDB is used as storage backend and Grafana to visualize the data. In this research, only InfluxDB was added to store the data and after each test the data was exported to file, which later allowed to compare the results between applications on a single chart.

2.5 Docker and Docker Compose

Docker and Docker Compose were used to simplify the development. This made starting all services that had be run together possible with only one command, eg. for django performance tests - django, postgres, k6 and influx. For every application a production ready Dockerfile was created. Additionally, Docker provides applications a layer of isolation from each other and the host.

2.6 PostgreSQL

For the Database Management System I chose PostgreSQL, which is the second most popular choice among database technologies

Chapter 3

Criteria description

3.1 Performance benchmark

3.1.1 Scenarios

The task for each application is to complete simple CRUD operations as fast as possible. For comparison of the performance of the applications, a few scenarios were developed:

- retrieving multiple objects (getMany)
- retrieving single object (get)
- updating a single object (put)
- creating a single object (post)
- deleting a single object (delete)

Scenarios were tested with a few different application loads, which are represented by a number of virtual users (VUs) - as mentioned in the k6 repository description they are glorified, parallel while(true) loops.

3.1.2 Database snapshots

To avoid any differences in the database between the tests, at the beginning of the script the database is populated and the snapshot is stored locally. Before each test, the snapshot is restored.

3.1.3 Application isolation

To be sure that the applications are running in an isolated environment, docker containers were used. To simplify the research, a Docker Compose configuration was prepared, that builds and starts all the necessary containers at once.

3.1.4 Test progress

The measures are gathered from each application in the following manner:

```
populate database;
store snapshot;
foreach test case do
    if application is running then
        kill application;
    remove volumes;
    if test case is not post then
        restore snapshot;
    start application;
    while application is not responding do
        wait for application;
    for 5 seconds do
        warmup requests;
    for 45 seconds do
        measured test;
    store result;
merge results;
```

3.1.5 Software versions and hardware

The tests were run on a laptop with the following specification:

Table 3.1: Hardware

Hardware	
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
RAM memory	16 GB @ 2400MT/s
Operating system	Ubuntu 20.04.2 LTS

Frameworks used to build the application were in the following versions:

Table 3.2: Framework versoins

Framework versions	
Django	3.1.4
ASP.NET	2.1.1
Express	4.17.1

3.2 Security

Chapter 4

System design

4.1 Database

For the tests, a database consisting of a single table was created.

4.2 Application

4.3 Environment

Chapter 5

Application implementation

5.1 Django

5.2 Express

5.3 ASP.NET

Chapter 6

Results

6.1 Performance

Table 6.1 shows an average of 95th percentile from the 10 tests mentioned in the previous chapters.

Table 6.1: Average p(95) response time in tests

AVERAGE z p(95)		filename		
test	concurrency	aspnet	django	express
delete	1	8.35	48.36	4.14
	8	9.75	143.84	10.94
	32	48.84	385.77	32.68
	128	192.80	1259.32	118.67
	512	675.12	4923.61	458.99
get	1	1.37	12.54	1.00
	8	6.42	54.46	5.98
	32	19.89	189.95	20.05
	128	84.64	643.71	79.42
	512	309.29	2605.04	321.64
getMany	1	48.16	56.93	24.14
	8	76.09	344.77	42.59
	32	177.36	1280.14	73.80
	128	325.66	4831.64	159.44
	512	683.23	19525.67	511.82
patch	1	9.13	45.58	5.20
	8	10.53	129.42	11.65
	32	50.86	358.92	35.65
	128	198.23	1237.40	130.69
	512	655.22	4806.19	501.05
post	1	6.73	40.73	5.13
	8	7.32	122.30	10.37
	32	42.36	321.98	29.24
	128	171.97	1069.47	120.47
	512	611.54	4185.22	407.89
put	1	9.12	45.20	5.26
	8	10.49	132.10	11.73
	32	51.82	359.56	34.78
	128	202.52	1236.24	129.99
	512	735.95	4817.58	507.11

6.2 Security

Chapter 7

Conclusion

Bibliography

[1] "Stack overflow 2020 developer survey." https://insights.stackoverflow.com/survey/2020#technology-web-frameworks. Accessed: 2021-03-14.

[2] J. K.-M. Adrian Holovaty, "The django book," 2007.

[3] "Django documentation - faq: General." https://docs.djangoproject.com/en/1.11/faq/general/#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names. Accessed: 2021-03-14.

List of Figures

List of Code Listings

List of Pseudocodes