Test Report

Test Report for U2.

Created by:

Petar Bakalov – 4634705

Contents

Versioning table

Version	Date	Description
1.0	20/06/2024	Initialization of document

Introduction

Testing is important part of every application, in order to ensure not only security of the application, but also making sure that the software solution works as intended when it's deployed in production.

For the context of "U2", I have implemented unit tests, to keep the development process smooth, and load tests, to simulate certain user behavior in production.

Unit Tests

The main purpose of unit tests is to test the smallest parts of the backend, to ensure that edge cases are included in the implementation of each method. Moreover, unit tests serve the purpose of "checkpoints" – whenever a new feature is implemented, the unit tests track if the old code has been broken by the new one.

Since semester 2, when unit tests had been introduced to me as a developer, they are a crucial part of every project of mine. However, with the new technologies that I learn, for example EFCore - type of ORM, some of the tests become unnecessary, as these new technologies handle a lot by themselves. I have found out that unit tests become more useful for methods, which implement more complex logic than CRUD.

Either way, I implemented a few unit tests in "U2" and certainly will implement more when the functionality of the application is extended. Moreover, unit tests are run through the CI/CD pipeline in my repository, generating a coverage report and pushing it to the 3rd party software "Codacy", which tracks the overall coverage of "U2" and any security leaks in the code itself.

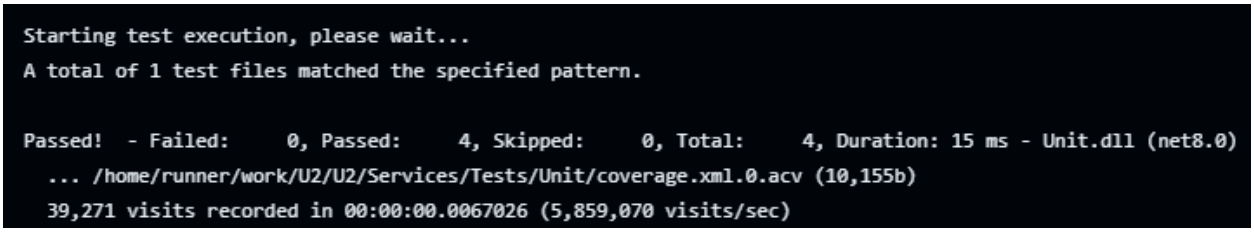

Fig. 1 "Screenshot of the logs from the pipeline, that runs the Unit Tests"

Load Tests

Load tests are part of performance testing, used to analyze how a system behaves under specific expected load. The primary goal of load testing is to identify performance bottlenecks and ensure that the system can handle the anticipated number of concurrent users.

I decided to use the tool "K6" to run a load test locally. My initial idea was to make use of the Fontys NetLab, to simulate actual production environment, however I could not set it up. The second option was to run the tests on my production cluster, as it is the actual production environment, but that would accumulate large financial cost, as my cluster is deployed in Azure. In Azure, I have enabled autoscaling for my application, which will create more instances of a service, if it is overloaded.

In the end, I decided to run the tests on my local cluster, even though the results will not be accurate, as there is no autoscaling and the speed of the executed methods also depends on the specs of my local machine.

The picture below shows the load tests I came up with:

```
export const options = {
    scenarios: {
        getAllUsers: {
            executor: 'ramping-arrival-rate',
            startTime: '30s',
            startRate: 50,
            timeUnit: '1s',
            stages: [
                { target: 100, duration: '30s' },
                { target: 100, duration: '3m30s' },
                { target: 0, duration: '30s' },
            ],
            preAllocatedVUs: 50,
            maxVUs: 100,
            exec: 'getAllUsers',
        },
        getAllVideos: {
            executor: 'ramping-arrival-rate',
            startTime: '30s',
            startRate: 500,
            timeUnit: '1s',
            stages: [
                { target: 1500, duration: '30s' },
                { target: 1500, duration: '3m30s' },
                { target: 0, duration: '30s' },
            ],
            preAllocatedVUs: 500,
            maxVUs: 1500,
            exec: 'getAllVideos',
        },
        getVideosForUser: {
            executor: 'ramping-arrival-rate',
            startTime: '30s',
            startRate: 500,
            timeUnit: '1s',
            stages: [
                { target: 1500, duration: '30s' },
                { target: 1500, duration: '3m30s' },
                { target: 0, duration: '30s' },
            ],
            preAllocatedVUs: 500,
            maxVUs: 1500,
            exec: 'getVideosForUser',
        }
    },
    discardResponseBodies: true,
};
```

Fig. 2 “Load test for U2”

I designed my load test to call multiple endpoints of the backend, to simulate a user behavior as much as possible. As you can see, the endpoint “getAllUsers”, which returns a list of every user in my application, has smaller number of virtual users calling it. The reason for this is that this endpoint is only called by the admins of the application, which, of course, are supposed to be fewer than the actual users of the application. Therefore, I decided to put significantly more virtual users for the other endpoints.

Analyzing the results

I ran the load test with different configuration – first with fewer users and then with more, to see the way my system performs and the differences between lower and higher load.

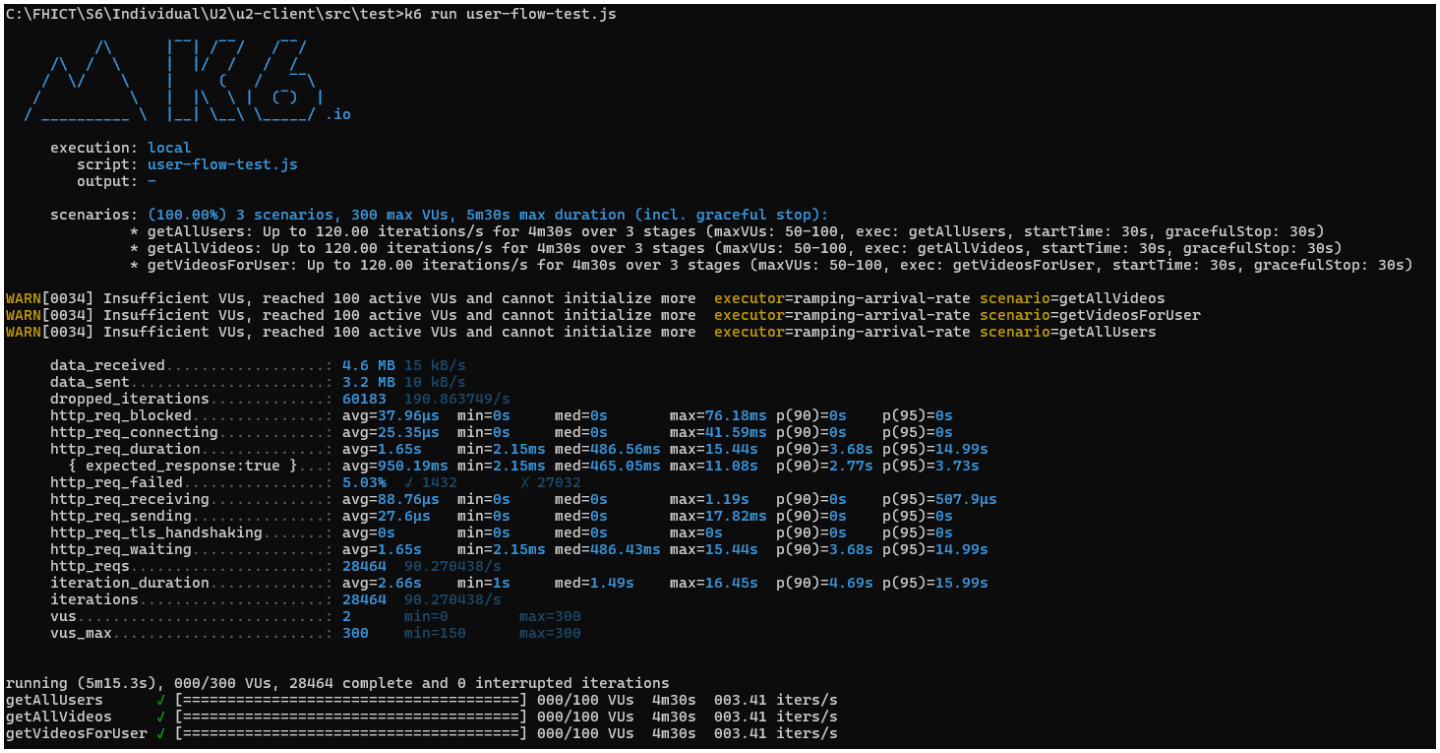

Fig. 3 “Load test of 300 total users”

If we look closer into the results of those tests, there are about 5% failed requests. The main reason for that is the MySQL database, which I use for user storage. MySQL only allows a certain number of concurrent connections, and if the cluster was in Azure, it would have created another instance of the pod, and the requests would have passed.

Looking at the average waiting time for a request it is 2.66 seconds. It seems like a lot, however that is boosted by the failed requests, which attempted to execute for almost 15 seconds each. If you look at the medium time for that data, it is around 490ms, which is acceptable.

Now let's take a look at the difference, when the load test is run with more users.

Fig. 4 "Load test of 1280 total users"

Again, looking at the percentage of failed requests, it is more than twice more than the previous configuration. However, keep in mind that the failed requests come from the fact that the tests are run on my local cluster and the request that fails is the one, used by admins. Technically the number of admins I have set is not realistic as for 1200 users, 80 admins is an overkill.

In conclusion, the load tests prove that the application will work as expected for 1280 users, using my local machine as a server and my local cluster with only two workers. The tests will certainly perform in a different way if they were in the NetLab environment, however the local tests are enough to prove that “U2” performs as expected with a large number of users.