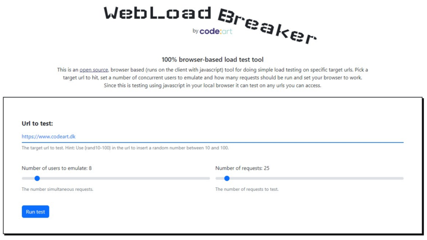
ALLAN THRAEN | 2 years ago | PDF |

Vision Demos & Prototypes Life as a Coder Tips and Tricks Website Improvements Frontend Development CMS

# WEBLOADBREAKER - BROWSER BASED WEB LOAD TESTING TOOL



100% browser-based load test tool

This is an open source, browser based (runs on the client with javascript) tool for doing simple load testing on specific target urls. Pick a target url to hit, set a number of concurrent users to emulate and how many requests should be run and set your browser to work. Since this is testing using javascript in your local browser it can test on any urls you can access.

Url to test:
https://www.codeart.dk
The target url to test. Hint: Use {rand10-100} in the url to insert a random number between 10 and 100.

Number of users to emulate: 8
The number simultaneous requests.

Number of requests: 25
The number of requests to test.

**WebLoadBreaker is a new, open source, browser based tool to help you do quick and easy load tests.**

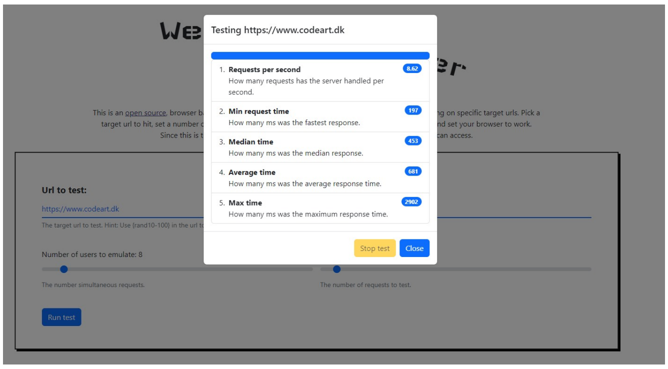
One of the key performance elements to test before your release your website is not just how fast the user experience is (which is often tested in tools like Lighthouse or PageSpeed), but also how efficient it behaves under a certain expected load. In order to do that test you need to put the site under a certain load and then measure both how many requests per minute it can process - but also how fast those requests respond.

There are numerous load testings tools available for this - I usually prefer Netling which is simple, but gets the job done. However, most of the the tools needs to be installed on an environment where they can access the urls that needs to be tested. This can sometimes be tricky and risky to do in a secure environment - not to mention the bureacracy sometimes needed to persuade IT to allow the tool.

To make things a bit easier I recently made **WebLoadBreaker** - a simple, browser based tool to do some simple load testing on a url. And of course it's available as **open source**.

WebLoadBreaker is really very simple. It uses vanilla js to fetch a provided url asynchroneously (and in parallel) a certain number of times, while simulating a certain number of users. While doing this it also measures the durection of each call and calculates some basic metrics on top of that.

Basically, you just provide it with a url to hit, a number of users and a total number of requests you want it to do - and then run the test.



Testing https://www.codeart.dk

1. **Requests per second** 8.62
   How many requests has the server handled per second.
2. **Min request time** 197
   How many ms was the fastest response.
3. **Median time** 453
   How many ms was the median response.
4. **Average time** 681
   How many ms was the average response time.
5. **Max time** 2902
   How many ms was the maximum response time.

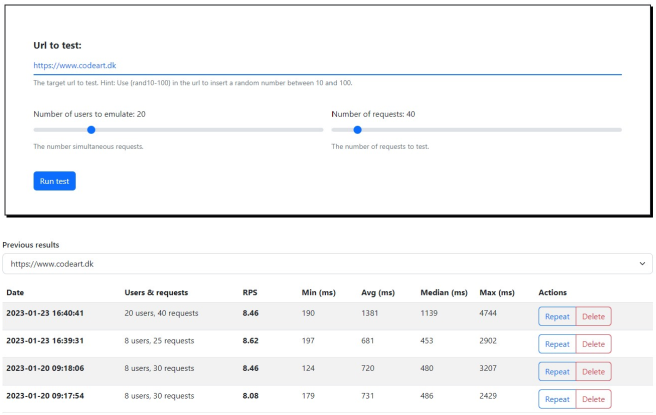
Url to test:
https://www.codeart.dk

Number of users to emulate: 8
The number simultaneous requests.
The number of requests to test.

The key metric I usually look for is Requests per second (RPS). As you can see above - this site, codeart.dk is not doing particularly well - however handling a lot of simultaneous users has never been a big need (at least not yet). It is of course also worth taking a look at the response times - keep in mind that with this tool you are typically only testing the html part - not the full page load with images, scripts, styles and so on. It should ideally always be under 1 second.

Another thing to keep in mind is that this is a tool running in your browser. The performance measured also includes roundtrip to the server and the capacity of the browser in handling multiple concurrent fetch requests. On well-performing sites you could potentially reach the browser/trafffic bottleneck before the servers. However, I have found that for many sites you can clearly see potential for improvements using this tool - and then when you improve them, you can also see the improvement with this tool. And that's really what it was made for.

After the test is complete, it will store the metrics in your browsers local storage so you can compare each url with previous results.



Url to test:
https://www.codeart.dk
The target url to test. Hint: Use {rand10-100} in the url to insert a random number between 10 and 100.

Number of users to emulate: 20
The number simultaneous requests.

Number of requests: 40
The number of requests to test.

Previous results
https://www.codeart.dk

| Date | Users & requests | RPS | Min (ms) | Avg (ms) | Median (ms) | Max (ms) | Actions |
|---|---|---|---|---|---|---|---|
| **2023-01-23 16:40:41** | 20 users, 40 requests | **8.46** | 190 | 1381 | 1139 | 4744 | Repeat Delete |
| **2023-01-23 16:39:31** | 8 users, 25 requests | **8.62** | 197 | 681 | 453 | 2902 | Repeat Delete |
| **2023-01-20 09:18:06** | 8 users, 30 requests | **8.46** | 124 | 720 | 480 | 3207 | Repeat Delete |
| **2023-01-20 09:17:54** | 8 users, 30 requests | **8.08** | 179 | 731 | 486 | 2429 | Repeat Delete |

All of this is done completely using javascript and can run in most modern browsers (tested in Chrome and Edge) without any problems.

You can even fetch the code from the repository and run it locally - that works as well.

*As I'm curious by nature - and want to see if this service ends up getting used, I put a Matomo-tracker in there - but it runs without cookies, anonymized IP's and it does not track the urls being tested. However, feel free to fork the code and remove the tracking - it's clearly marked.*

## Additional Features

- You can set the parameters in the URL like this: https://codeartdk.github.io/WebLoadBreaker/?url=https://www.codeart.dk/&users=5&requests=10 and thereby link to a specific test.
- You can define random numbers in the URL - for example to bust caches when testing. Like this: https://www.codeart.dk/?nocache={rand0-1000}. This will insert a random number between 0 and 1000 where the placeholder is.
- It is a progressive web application. This means you can install it on your desktop and get a shortcut for it for easy access - or have it as an app on your smartphone (although I don't know why you'd want to do that).

## Shortcomings

- The biggest problem right now is that the script cannot see the HTTP response code returned from the fetch operations. This is due to CORS and is a security feature built into the browsers. This means that you have no way of knowing if every single hit is a 200 Ok, or a 'Server busy' or maybe even a 404. I'm still investigating ways to get around this issue - but for now my best advice is to take a look in the browsers Developer Tools in the "Network" pane where all the requests and responses can be seen - if you suspect that something is wrong. Typically server rejections are a lot faster than normal processing - so if you all of a sudden get responses back way faster than before - then take a look.
- There is very little input validation still. This is a work in progress. If you input garbage you'll get garbage out.

## Call for participation

If you like the project and would like to make it even better - either by submitting pull requests, bug reports or suggestions for features I'd really appreciate it. Drop a comment below or send mea  mail - or fork the project and make a pull request!

**https://codeartdk.github.io/WebLoadBreaker/**

**https://github.com/CodeArtDK/WebLoadBreaker**

Try it out!

Vision Demos & Prototypes | Life as a Coder | Tips and Tricks | Website Improvements | Frontend Development | CMS