

# Research Document

## Applying Compression on the Web applications

Eindhoven, October 20<sup>th</sup>, 2021

# Table of Contents

Revision History: .....	2
Introduction.....	3
What are the main problems when developing software solution ? .....	3
What are the DOT methods for developing software solution ? .....	3
The solution for developing software solution.....	4
How to solve the project size problems when using libraries .....	4
What is the compression ? .....	4
How we can apply the compression on web applications .....	5
Performance Tests .....	5
What are the benefits we get from the Compression ? .....	8
References .....	8

## Revision History:

Date Changed	Document Version	What was changed
20/10/2021	1	
26/11/2021	2	Adding Main Questions and DOT methods

## Introduction

Recently, applications have become more sophisticated both the implementation and architectures have been used, make the applications big in size for downloading and installing, taking time consumption and make uncomfortable to clients who used the applications. For this reason, this research document will focus on the topic of compression, a solution to make the application processing faster, give the satisfaction to the clients.

## What are the main problems when developing software solution ?

Time and cost budget are the main point when comes to build project. In order to keep the time for the project, we may have to consider hiring more employees (as Human Resources) to participate in the project, means the cost budget will be exceeded, that is why we need different strategies.

## What are the DOT methods for developing software solution ?






Page	Discussion				
Methods					
					
Library	Field	Lab	Showroom	Workshop	Extra
Available product analysis	Document analysis	A/B testing	Benchmark test	Brainstorm	Joker
Best good and bad practices	Domain modelling	Component test	Ethical check	Business case exploration	
Community research	Explore user requirements	Computer simulation	Guideline conformity analysis	Code review	
Competitive analysis	<a href="#">Focus group</a>	Data analytics	Peer review	Decomposition	
Design pattern research	Interview	Hardware validation	Pitch	Gap analysis	
Expert interview	Observation	Non-functional test	Product review	IT architecture sketching	
Literature study	Problem analysis	Security test	Static program analysis	Multi-criteria decision making	
SWOT analysis	Stakeholder analysis	System test		Prototyping	
	Survey	Unit test		Requirements prioritization	
	Task analysis	Usability testing		Root cause analysis	

Figure 1 The DOT methods [4]

From the pictures above, my choices for this document will mostly in the “Library” section as this topic have been solved for a long-term and continue until now. One of the main reasons finding solutions for improving the efficiency of developing software solution sill continue today, because these are optimization problems, means it will not have a final solution, means have rooms for other better solutions. I choose “Available product analysis” with “Community research” with the “Observation” and “Benchmark Test” as for the conclude of my suggested solutions.

## The solution for developing software solution

In order to balance the cost budget and time, using the third-party solutions is the recommended options when considered to develop IT projects. The main benefits are third-party solutions have already built to solve the problems that they are intended to, means as for the consumers (here the developers) just learn how to use it and apply it in the projects, without how to know the details inside.



Figure 2 Cooperation between Philips and AWS for healthcare solutions [3]

An example that Philips have cooperated with Amazon to use AWS services and infrastructures, more detail is Philips Healthcare platform live on AWS services, so that Philips can focus on their solutions without thinking on the hardware or infrastructure, which handled by AWS.

For the software scopes, developers will have to use third-party libraries on the markets like Nuget, NPM, YARN, Gradle, Maven, ... to develop solutions as fast as possible. Solutions may use quite a lot of libraries and we have the package managers to keep track libraries that have been installed for the projects. However, a problem will raise that the more libraries we use, the more size of applications that can take amount of hard memory for storage or for downloading to the users.

## How to solve the project size problems when using libraries

The first solution is to import necessary part of libraries that we only to use for the project, however it will take a lot of time to look the source code and refactor the code by removing unnecessary parts of libraries, so we come to the second solution, which is using compression.

## What is the compression ?

In data compressing, compression is the process of encoding data using fewer bits than the original without losing the data when decompressing. The compression will trade off between the size of data and the time for processing the data. It means, when we compress the file, we will have smaller size of files that suitable to transmit over protocols, but it will take time than usual as to have including total time of compressing and decompressing the file.

## How we can apply the compression on web applications

We have used compression on file using the compressing software like WinRAR, Windows built-in compressing software so that we can be able to submit the file to the internet easily. If we observe the website looks like, it is consisting of the code implementation of HTML/CSS/JS and static resources like image, sounds, videos, which can compress, so it means web applications can be compressed without problems. So, we can compress response when clients send a request to server and let the clients decompressing the response automatically.

## Performance Tests

I used NodeJS as the backend framework for the purpose of testing the performance, my assets will be the static website version of Bamboo restaurant.

During the test, I open the Developer Tools panel (pressing F12), to observe the process and time performance of my approach, make a comparison of scenarios.

For the first version, without the compression, my backend server implementation only has one API function call, which return a completely website to my browser.

A screenshot of a code editor window titled 'JS index.js'. The code is written in JavaScript and implements a simple web server using Express.js. It includes line numbers from 1 to 13. The code sets up an Express app, uses the 'path' module, and defines a single GET route for the root path that serves a file named 'index.html' from the current directory. The server listens on port 3000.

```
JS index.js > ...
1  const express = require('express');
2  const app = express();
3  const path = require('path');
4  const router = express.Router();
5
6  router.get('/', function(req, res)
7  {
8    |   res.sendFile(path.join(__dirname+'/' + 'index.html'));
9  });
10
11 app.use('/', router);
12 app.use('/assets', express.static(path.join(__dirname, 'assets')));
13 app.listen(process.env.port || 3000);
```

Figure 3 The first version implementation without compressing

After that, build and run my backend server, make a call to the APIs, and I get the website as expected.

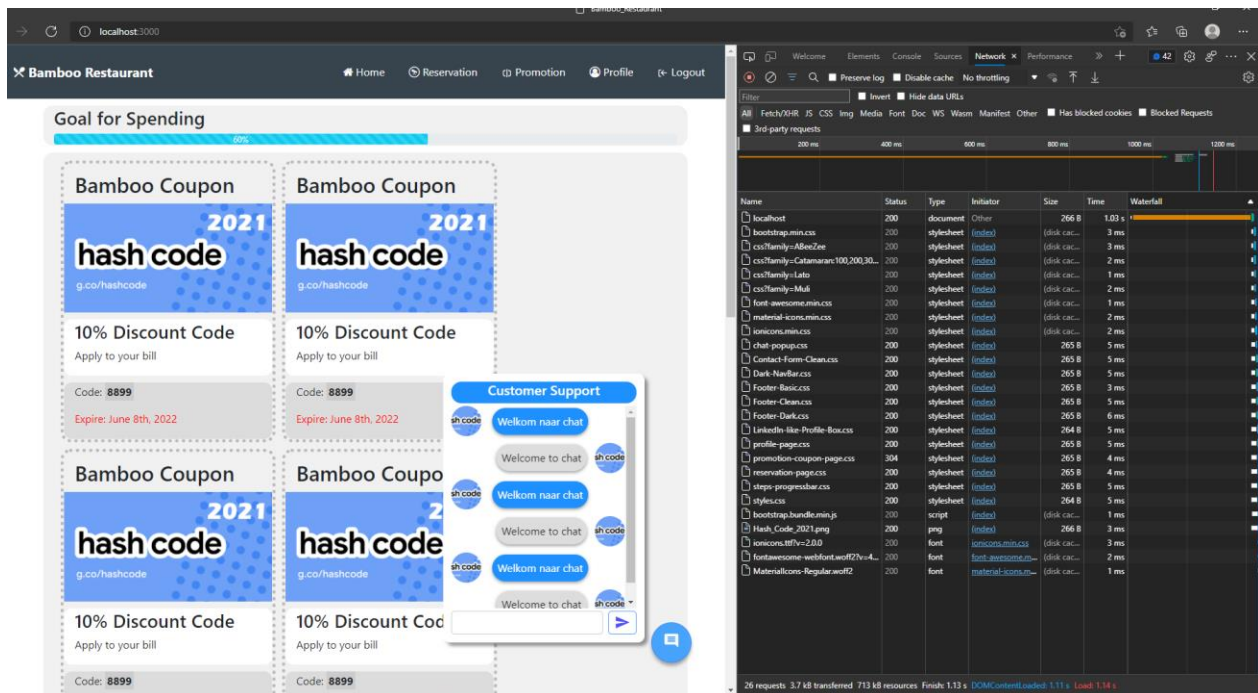


Figure 4 The stack trace of loading content (1st version)

As you can see, we also get the detail how my website has been loaded to the browser and the time indication for my website loaded.

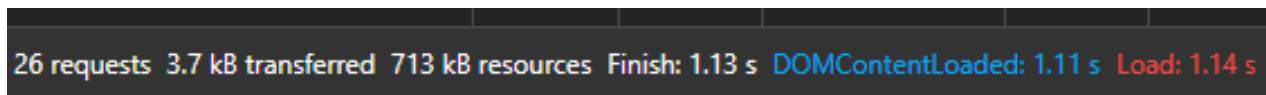


Figure 5 The performance details for first version

Zoom to the overview information at the bottom of developer's tools panel, we can see that my website takes 1.14 second to finish the loading (or 1140 Ms), which is acceptable to clients as it is loaded below 2 seconds, but can we improve it better.

So, for the next test, with the same website, but my backend server implementation will include the compression library and it only add 2 lines of code compared to my initial implementation.

```

JS index.js > ...
1  const express = require('express');
2  const app = express();
3  const path = require('path');
4  const router = express.Router();
5  const compression = require('compression');
6
7  router.get('/',function(req,res)
8  {
9    res.sendFile(path.join(__dirname+'/index.html'));
10 });
11
12 // Compress all HTTP responses
13 app.use(compression());
14 app.use('/', router);
15 app.use('/assets', express.static(path.join(__dirname, 'assets')));
16 app.listen(process.env.port || 3000);

```

Figure 6 The second version implementation with compressing

After that, I build and run for my updated implementation, make a call to the APIs, I still get the website as I expected.

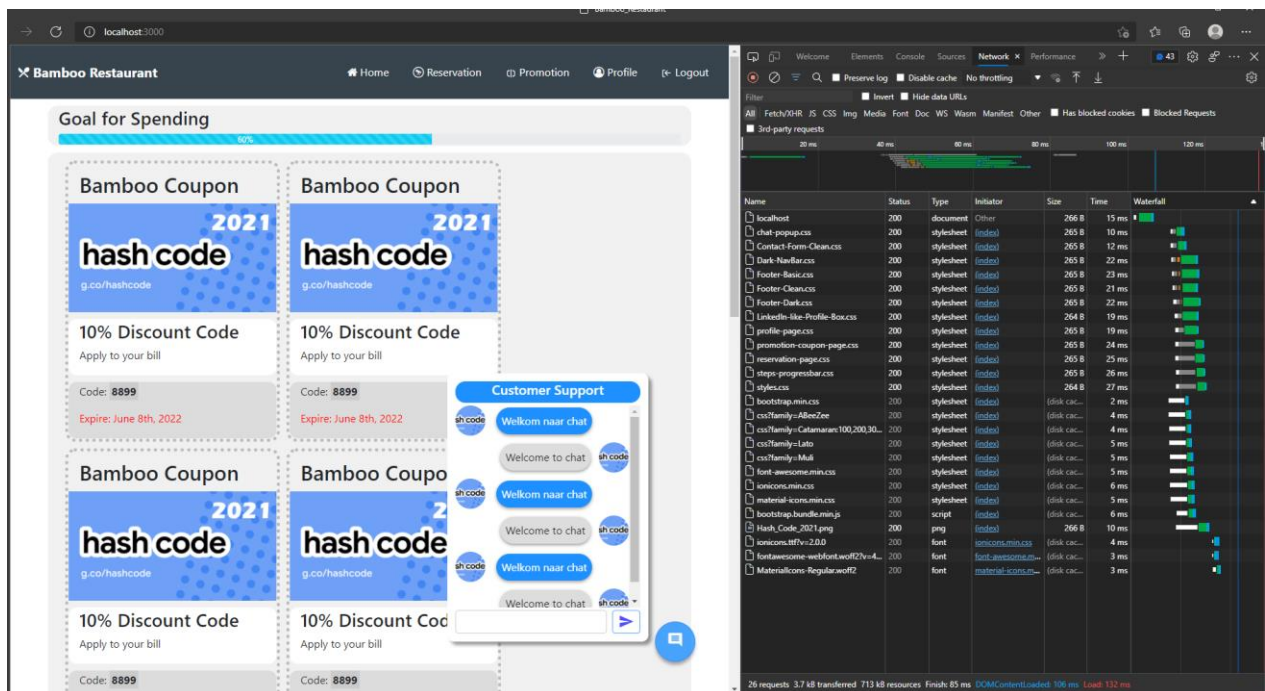
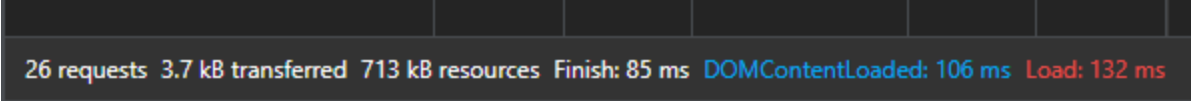


Figure 7 The stack trace of loading content (2nd version)

However, the interested part is the time performance of the compressed version, as you can see, my website takes 132 ms to finish loading the website.



26 requests	3.7 kB transferred	713 kB resources	Finish: 85 ms	DOMContentLoaded: 106 ms	Load: 132 ms
-------------	--------------------	------------------	---------------	--------------------------	--------------

Figure 8 The performance details for second version

Compared to the previous without compression, the loading time is 90% less than, which is an impressive result.

## What are the benefits we get from the Compression ?

From the result of my benchmark tests, I can see the opportunities of applying the compression to the website application by reducing the size of code implementation, means more technologies/libraries can be included. It is also benefited to build a Single Page Application, so at the beginning clients already have a complete application, when transmitting /querying the data, only the necessary data will be considered, reduce the workload for server.

## References

1. Rico, V. V. (2020, June 26). *GZIP compression with node.js*. Medium. Retrieved October 25, 2021, from <https://medium.com/@victor.valencia.rico/gzip-compression-with-node-js-cc3ed74196f9>.
2. Wikimedia Foundation. (2021, October 7). *Data compression*. Wikipedia. Retrieved October 25, 2021, from [https://en.wikipedia.org/wiki/Data\\_compression](https://en.wikipedia.org/wiki/Data_compression).
3. National Council on Vocational Education. (1991). *Solutions*. Amazon. Retrieved November 29, 2021, from [https://aws.amazon.com/solutions/case-studies/philips-reinvent/?did=cr\\_card&trk=cr\\_card](https://aws.amazon.com/solutions/case-studies/philips-reinvent/?did=cr_card&trk=cr_card)
4. Methods. (n.d.). Retrieved November 29, 2021, from <https://ictresearchmethods.nl/Methods>