Bachelor Thesis Project

Improving load time of SPAs

- An evaluation of three performance techniques

Author: Rasmus Eneman
Supervisor: Tobias Olsson
Semester: VT/HT 2016
Subject: Computer Science

Abstract

The code size of single page web applications are constantly growing which do have an negative effect on the load time. Previous research have shown that load time are important to users and that a slow application will lose potential customers even before it has loaded. In this paper three architecturally far-reaching techniques are measured to see how they can improve the load time and help to decide if an application should be built with one or more of the tested techniques which are HTTP2 push, Code Splitting and Isomorphism. The experiment shows that Isomorphism can provide a big improvement for the time to first paint and that Code Splitting can be a useful technique for large code bases on mobile phones.

Keywords: SPA, load time, first meaningful paint, HTTP2 push, Code Splitting, Isomorphism

Contents

1 Introduction

1.1 Background

Single page web applications (SPAs) [1] are web applications that after the initial load lives only in the browser. If they need data they can request data using HTTP or Websockets but the request is driven by the application, not the browser. These kind of applications rely on JavaScript as they need the ability to render results of user actions and be able to perform every action without loading a new page. The additional JavaScript size lead to additional transfer, parsing and execution time which affect the load time of the page, but may after load do actions without the network.

1.2 Previous research

According to Miller [2], the load of an application should not exceed two seconds due to limitations of humans short term memory, Nielsen [3] however argues that one second is the maximum for humans to focus uninterrupted on the task.

To find out how these numbers apply to web page load times, Fiona [4] conducted a series of experiments where she saw that 26-31% of the users stopped waiting for a website to load after two seconds and after having previously visited long-loading web pages 12-22% stopped waiting already after one second. This shows how important the load time is in terms of market value, risking to lose 10-30% of the potential customers due to slow load times is expensive.

Previous research [5] have also shown that usage patterns for mobile browsing is different than on normal PCs. On mobile most of the pages visited are visited for the first time (60%) and only 25% of all pages are visited more than twice which mostly abolish the possible gains of browser caching. This further strengthens the need for a better initial load time as techniques such as caching loses much of its effect if more than half of the visits are for the first time.

1.3 Problem formulation

As the load time is important for users, it has a role in how successful a product will be and should therefore be part of the product development. Many techniques for improving load time are non trivial to implement and affect architecture or technical decisions that usually need to be decided upon early on or else require large refactorings of the code and architecture.

1.4 Motivation

The total JavaScript (JS) size of web pages are constantly growing [6], which affects the load time of the page, partly because more data has to be transferred to the browser and partly because the browser needs to parse and execute more code before the page can be rendered. At the same time web browsing is shifting towards less capable devices like mobiles and tablets [7]. However before investing time or making important architectural decisions a company wants to know that they will archive the desired results.

1.5 Research Question

RQ1	To what degree does architecturally far-reaching techniques that aim to improve load time of SPAs, affect load time of SPAs?

1.6 Scope/Limitation

There are many possible techniques to improve the load time and researching all of them is not possible for the set timeframe of this research. Three different techniques have been selected for this research to have enough time for thorough evaluation of their impact.

The techniques have been chosen because they are affecting the architecture and sometimes the possible selection of other techniques of the application. Another reason is that they are the three most specific techniques that are often discussed for load time optimization of large SPAs. Others are usually more theoretic and require investigation and knowledge of the application to understand how they affect the architecture and what possible savings they may provide, which would not have given a general understanding that can be applied to other applications.

1.6.1 Code splitting

Code splitting allows the server to deliver the JavaScript in smaller parts, for example is it possible to only load the JavaScript needed for the first view in the initial page load and then download the rest of the JavaScript after the application has been loaded.

1.6.2 Isomorphic

An isomorphic web application have support for running both on the client and the server. This makes it possible to render the first, not cached, request on the server and provide everything needed to display the initial page in the first HTML response. In a pure client side application the browser would first have to download the HTML to find the JavaScript, download the JavaScript and then execute that JavaScript before being able to render at all.

1.6.3 HTTP2 push

HTTP2 have support for pushing resources before the client have requested them. This allows for the server to push resources necessary to render the page, for example CSS, fonts and JavaScript directly after the browser has requested the HTML without waiting for the browser to finish downloading and parsing the HTML to then request the other resources. In theory this could reduce the load time by one or more Round Trip Times (RTTs), the time it takes for a single TCP package to travel from the client to the server and back. Netravali has shown that stacked up RTTs are a big part of the total load time of web applications [8].

1.7 Target group

The target is professional web developers that develops large SPAs, as all of them will have to weigh the time spent on optimization versus the potential gains.

2 Method

2.1 Scientific Approach

A controlled experiment that collects quantitative data will be used to answer the research question.

Depending on the measurement the expected outcome is either positive or negative which is described later in the method description.

2.2 Method Description

The three mentioned techniques in **1.6** will be implemented in a web application. An experiment is then conducted where each of the techniques are tested to see how they affected the load time of the web application. Both the time to the first meaningful paint and the time it takes until the application is responsive to input are measured.

First meaningful paint is the first frame that contain meaningful information to the user. This could for example include buttons and menus so the user can orient him or herself, important information like headers, or other things that can make the page usable even before it has been fully loaded. There is no formal definition of what is included in the first meaningful paint, instead it is something that must be judged on a case-by-case basis.

The experiment is performed in the Google Chrome web browser and the measurements is done using the Chrome developer timeline tool.

Two different platforms will be used, a modern laptop over a fast wifi connection and a modern phone over a mobile connection. To avoid big differences in the measurements due to noise, the actual network will be a local web server running on the laptop and for the phone measurements the network will be tunneled over a USB cable and then the bandwidth and the latency is emulated using the standard settings in the Chrome developer tool for wifi and good 3g with 30 Mb/s 2ms RTT for the wifi and 1 Mb/s 40ms RTT for the mobile connection.

The laptop is an Asus UX32LN with 12 GB RAM and Ubuntu 15.10, the phone is a Nexus 5 with Android 6.0. Both runs Chrome version 48

The application displays dashboards with a set of different widgets, the first meaningful paint is defined as the first frame which contain the menu, and the all widgets but not necessarily the data in those widgets. This allows the user to either discover where the data will be presented when available or find the menu item if they instead want to perform an action. Due to the design of the application the data can not be included in the first meaningful paint as it may not exist until an external event has occurred.

Responsive to input is defined as the point where all event handles for all buttons visible in the first usable paint has been bound.

The first load of the page is not cached and then a second measurement is performed for the time it takes to navigate from the first dashboard tab to the second which is cached. When code splitting is used, the navigation requires the application to download additional JavaScript to render the new widgets.

The JavaScript size of the application is 2.6 MB uncompressed, but all transferred data will be gzip compressed which makes the transferred size less than 2.6 MB. For the code splitting tests the size of the main application is 2.1 MB and 0.49 MB is downloaded during the navigation. To see how code splitting affects larger applications as well an additional non optimized test run is performed with 1.3 MB additional JavaScript added, for a total of 3.9 MB JavaScript. Additional navigation tests for the extra size will not be performed because how the application needs to load the split JavaScript depends on the individual application and a generalized measurement holds no value.

The independent variables that are purposely modified in the experiment is which techniques are enabled, on which platform (device, bandwidth, RTT) the application is run and the code size. The dependent variable is whenever or not a navigation requires extra code to be loaded which is true if code splitting is enabled. Every configuration of the application is measured 20 times each for the three described measurements of time to first meaningful paint, time to fully loaded and time to navigate.

The results will be compared to their corresponding baseline, which is the non optimized version on each platform using a Wilcoxon rank-sum test [9] with significance level (α) = 0.05 which is a nonparametric statistical hypothesis test to see whenever the result show a statistically significant difference, or not. Wilcoxon rank-sum is used instead of a t-test as it does not require the assumption of normally distributed data.

2.2.1 Propositions

For all measurements of code splitting a positive result is expected.

The outcome for both the time to first meaningful paint and the time for fully loaded is expected to be improved but that the time to navigate is expected to be worse.

For the isomorphic version a positive result is expected for the time for first meaningful paint with an improved load time but the expected result for time to fully loaded and navigation is negative, with no significant difference.

For the version with HTTP2 push a positive result is expected for both the time to first meaningful paint and the time to fully loaded with improved times and the expected result for time to navigate is negative with no significant difference.

2.3 Reliability and Validity

Due to time restrictions as well better developer tools for performance measurements than in other browsers, the experiment was only performed in Chrome. Other browsers may perform differently which affects the validity of the experiment. While the network bound part of the load would be equal, there might be important differences in parse, execution and render time.

This is an external validity issue as the results may be different in other browsers but it should not affect the internal validity or reliability of the result. If other browsers had been used a reliability issue might be introduced in the experiment as it is not possible to exactly determine when the frame containing the first meaningful paint have been rendered in other browsers because the reported data is not as detailed.

Another external validity issue that is due to time restrictions is that the techniques only is implemented and measured in a single application. The chosen techniques are not application specific but their impact may still vary from application to application.

3 Implementation

The techniques are implemented in an existing React application to get a realistic code size and behavior.

HTTP2 push is implemented using the Node.js http2 library based on the provided example code with the library. While this technique does not affect the application on a global scale, it requires you to implement your own HTTP server using one of the available HTTP2 libraries. The server must also have knowledge of which files, other than the requested one, should be sent to the client. For the purpose of this experiment this is based on a hard coded list in the server, but maintaining that list in a real life scenario might be painful and more advanced solutions might be needed.

Code splitting is implemented using Webpack. Webpack is a build tool that tracks the files of the project and creates a dependency graph so that it knows which files every file depend on. By using this knowledge it will only output a bundle with the files that are actually imported by either the entry point, a file that is imported from the entry point, a file imported by one of those files and so on. If a file can not be reached, it is not included.

You can also add a special reference to a file which means that Weback will not include that file in the main output but instead build it, and its dependency graph, into a different output file that can be included at runtime instead.

When using this technique you must ensure that all references to the files that you want to split into a different output is replaced with these special references and that all code accessing objects and functions in the split out files is guarded so that it does not try to access the objects and functions before first having loaded the other file as well. How big impact a refactor to split out parts of the code is depend on how many places the split out code is accessed from.

When isomorphism is used there are two big changes needed to the application, first all code that is rendering something on screen must support rendering to HTML as well as DOM elements and second every access to a browser feature like media APIs, storage, network access and more must be replaced with equivalent solutions on the server.

React has built-in support for server side rendering which for most cases solves the first problem for React based applications.

For the tested application, network access is only used for receiving data which is replaced by database access when on the server, buttons with click-handlers is rendered as links and browser storage access is replaced with database calls.

4 Results

Table 4.1 shows which platform and which technique is tested in each configuration of the application. Each configuration is given a sample number to make it easier to keep track of all configurations.

Sample	Platform	HTTP2	Code splitting	Isomorphic	Extra size
1	Laptop				
2	Laptop	Y			
3	Laptop		Y		
4	Laptop			Y	
5	Laptop	Y	Y	Y	
6	Mobile				
7	Mobile	Y			
8	Mobile		Y		
9	Mobile			Y	
10	Mobile	Y	Y	Y	
11	Laptop				Y
12	Mobile				Y

Table 4.1: Sample definition

4.1 First meaningful paint

Table 4.2 shows a summary of the time it takes until the first meaningful paint is displayed on the screen. The full raw data is too big to include directly in this section so it is instead available as Appendix 1.

The mean and median values are displayed in milliseconds. Standard deviation and standard error of the mean gives an idea of how varied the data is without displaying the raw values.

Sample	Number of samples	Mean	Standard deviation	Standard error of the mean	Median
1	20	1440.150	291.763	65.240	1508.500
2	20	2279.800	127.263	28.457	2242.500
3	20	1354.600	256.080	57.261	1502.500
4	20	601.400	87.489	19.563	632.500
5	20	978.800	195.282	43.666	893.500
6	20	4276.100	251.504	56.238	4217.000
7	20	6169.600	369.249	82.567	6262.500
8	20	3995.550	236.664	52.920	3963.500
9	20	631.800	94.277	21.081	633.000
10	20	5443.400	334.359	74.765	5471.000

Table 4.2: Time to first meaningful paint summary

4.2 Fully loaded

Table 4.3 show a summary of the time it takes until the application is fully loaded. The times are again displayed in milliseconds.

Sample	Number of samples	Mean	Standard deviation	Standard error of the mean	Median
1	20	1440.150	291.763	65.240	1508.500
2	20	2279.800	127.263	28.457	2242.500
3	20	1354.600	256.080	57.261	1502.500
4	20	1500.850	245.328	54.857	1585.500
5	20	2400.250	251.687	56.279	2302.500
6	20	4391.950	187.272	41.875	4338.500
7	20	6256.700	366.579	81.970	6345.500
8	20	4165.700	267.517	59.819	4239.000
9	20	4345.700	316.195	70.703	4267.000
10	20	5539.900	338.087	75.598	5563.000
11	20	1518.900	14.955	66.879	1502.000
12	20	5506.000	29.346	131.238	5519.000

Table 4.3: Time to fully loaded summary

4.3 Navigation time

Table 4.4 show a summary of the time it takes to fully load the second page of the application from that the menu button has been clicked. The times are again displayed in milliseconds.

Sample	**Number of samples**	**Mean**	**Standard deviation**	**Standard error of the mean**	**Median**
1	20	421.000	221.255	49.474	308.500
2	20	414.050	179.749	40.193	312.500
3	20	520.250	169.489	37.899	430.000
4	20	485.900	247.206	55.277	325.500
5	20	820.900	681.077	152.293	430.000
6	20	841.750	126.034	28.182	811.500
7	20	813.700	74.039	16.556	810.500
8	20	1372.050	158.679	35.482	1317.500
9	20	682.050	66.098	14.780	680.000
10	20	1119.150	64.130	14.340	1110.000

Table 4.4: Time to navigate summary

5 Analysis

In this section I will display boxplots and visualize the result but explanations, possible reasons and conclusions will be discussed in the next section. After the results from the two platforms have been displayed, a table that displays the result of a Wilcoxon rank-sum test and answers whenever the each result shows a statistically significant difference from the baseline.

The Wilcoxon rank-sum test is chosen because the result does not follow a generic probability distribution and therefore a nonparametric test had to be used.

5.1 First meaningful paint

5.1.1 Laptop

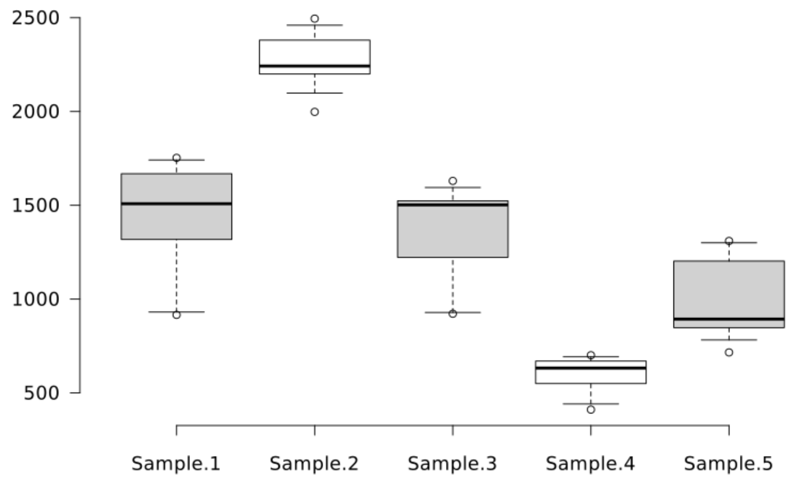

Figure 5.1: Boxplot of time to first meaningful paint on laptop

Figure 5.1 display the time it takes until the first meaningful paint is displayed on the screen for the different configurations measured on the laptop. The values are in milliseconds.

Sample 1 is the baseline without any techniques to improve the performance enabled. Sample 2 is using HTTP2 push and is quite a bit slower and only Sample 4, which is using the isomorphic capability is considerably faster.

5.1.2 Mobile

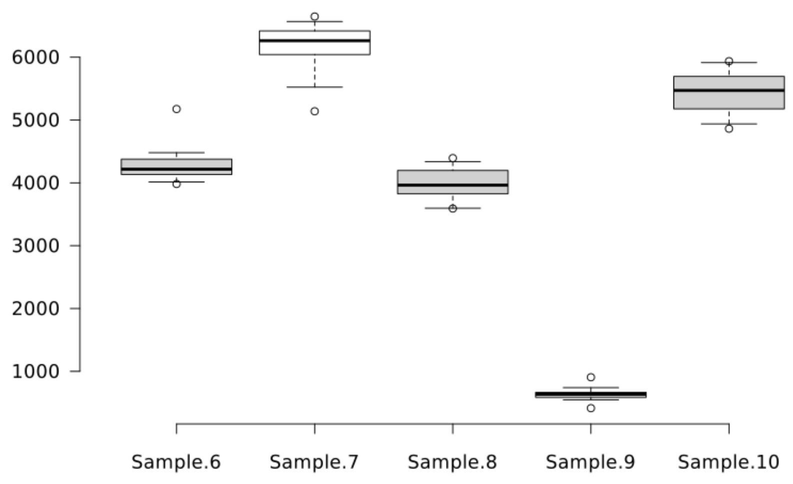

Figure 5.2: Boxplot of time to first meaningful paint on mobile

Figure 5.2 display the time it takes until the first meaningful paint is displayed on the screen for the different configurations measured on the mobile phone. The values are in milliseconds.

Sample 6 is the baseline without any techniques to improve the performance enabled. Sample 7 is using HTTP2 push and is quite a bit slower. Sample 8 is using code splitting and is slightly faster than the baseline while sample 9, which is using the isomorphic capability is considerably faster

5.1.3 Statistical Analysis

Did the techniques show a difference that are statistically significant? The p values is calculated using a Wilcoxon rank-sum test with significance level (α) = 0.05 where each of the techniques is compared to the corresponding non optimized version. So sample 2, 3, 4 and 5 is compared to sample 1 and sample 7, 8, 9 and 10 is compared to sample 6.

Sample	p value	Statistical significance
2	p < 0.001	Y
3	p = 0.199	N
4	p < 0.001	Y
5	p < 0.001	Y
7	p < 0.001	Y
8	p = 0.001	Y
9	p < 0.001	Y
10	p < 0.001	Y

Table 5.1: Statistical analysis of first meaningful paint results

5.2 Fully loaded

5.2.1 Laptop

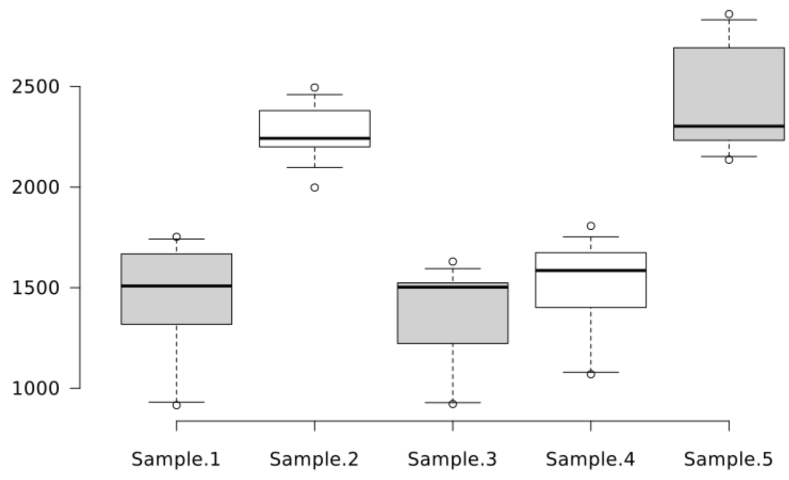

Figure 5.3: Boxplot of time to fully loaded on laptop

Figure 5.3 display the time it takes until the application has been fully loaded for the different configurations measured on the laptop. The values are in milliseconds.

Sample 1 is the baseline without any techniques to improve the performance enabled. Sample 2 is using HTTP2 push and is quite a bit slower. Sample 4 that is using the isomorphic capability were considerably faster in the first meaningful paint measurements but does now show the same result as the baseline.

5.2.2 Mobile

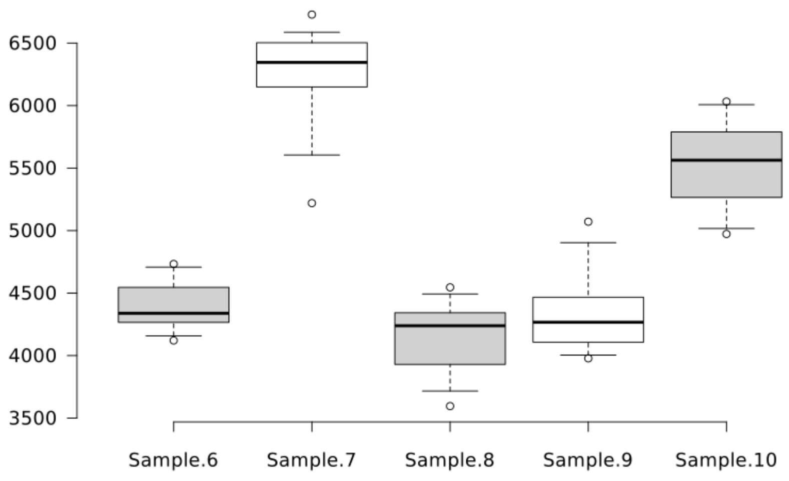

Figure 5.4: Boxplot of time to fully loaded on mobile

Figure 5.4 display the time it takes until the application has been fully loaded for the different configurations measured on the mobile phone. The values are in milliseconds.

Sample 6 is the baseline without any techniques to improve the performance enabled. Sample 2 is using HTTP2 push and is quite a bit slower. Sample 8 is using code splitting and is a bit faster than the baseline. Sample 9 that is using the isomorphic capability were considerably faster in the first meaningful paint measurements but does now show the same result as the baseline.

5.2.2.1 Code splitting

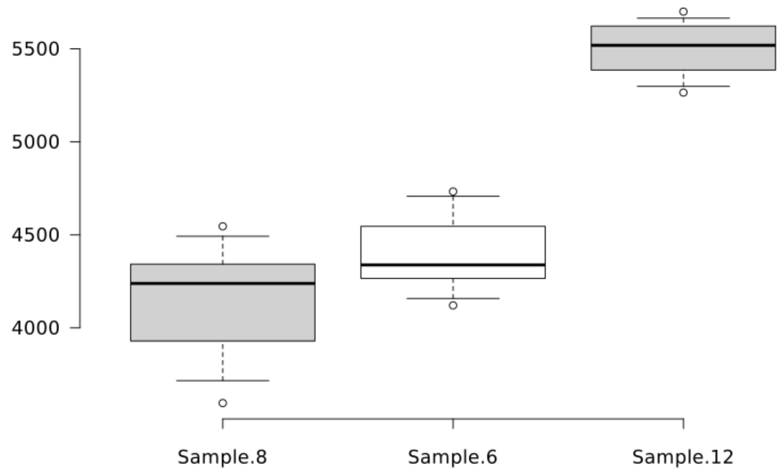

Figure 5.5: Boxplot of time to fully loaded on mobile with different code sizes

Figure 5.5 show how different code size affect the time to fully loaded on the mobile phone. Sample 8 is using code splitting to reduce the size to 2.1 MB, sample 6 is the baseline with no techniques to improve the performance activated on 2.6 MB and sample 12 is the version with extra code size on 3.9 MB.

5.2.3 Statistical Analysis

Did the techniques show a difference that are statistically significant? The p values is calculated using a Wilcoxon rank-sum test with significance level (α) = 0.05 where each of the techniques is compared to the corresponding non optimized version. So sample 2, 3, 4, 5 and 11 is compared to sample 1 and sample 7, 8, 9, 10 and 12 is compared to sample 6.

Sample	p value	Statistical significance
2	$p < 0.001$	Y
3	$p = 0.199$	N
4	$p = 0.473$	N
5	$p < 0.001$	Y
7	$p < 0.001$	Y
8	$p = 0.001$	Y
9	$p = 0.298$	N
10	$p < 0.001$	Y
11	$p = 0.152$	N
12	$p < 0.001$	Y

Table 5.2: Statistical analysis of fully loaded results

5.3 Navigation

5.3.1 Laptop

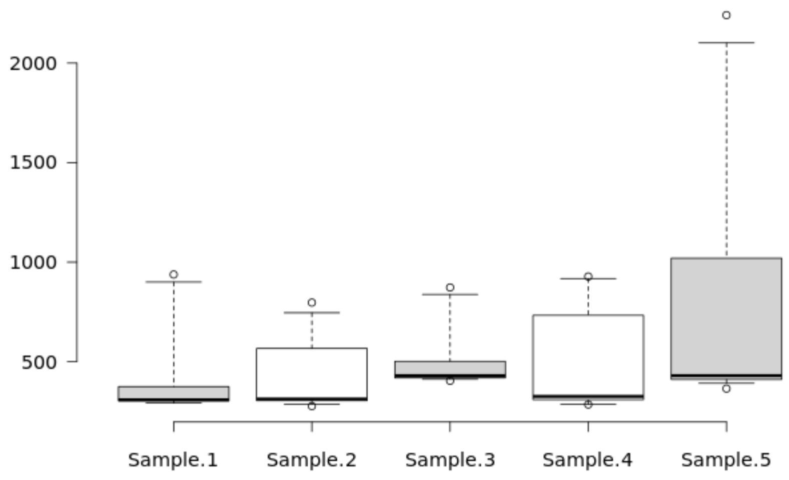

Figure 5.6: Boxplot of time to navigate on laptop

Figure 5.6 display the time it takes until the next page of the application has been fully loaded from when the menu button is clicked on the laptop. The values are in milliseconds.

Sample 1 is the baseline without any techniques to improve the performance enabled. Sample 3 and 5 is using code splitting and is a bit slower as additional code for the next page needs to be downloaded.

5.3.2 Mobile

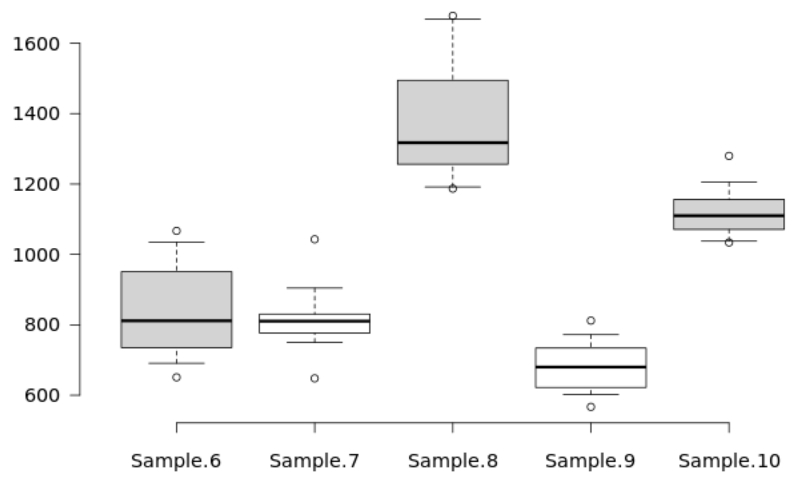

Figure 5.7: Boxplot of time to navigate on mobile

Figure 5.7 display the time it takes until the next page of the application has been fully loaded from when the menu button is clicked on the mobile phone. The values are in milliseconds.

Sample 1 is the baseline without any techniques to improve the performance enabled. Sample 3 and 5 is using code splitting and is a bit slower as additional code for the next page needs to be downloaded.

5.3.3 Statistical Analysis

Did the techniques show a difference that are statistically significant? The p values is calculated using a Wilcoxon rank-sum test with significance level (α) = 0.05 where each of the techniques is compared to the corresponding non optimized version. So sample 2, 3, 4 and 5 is compared to sample 1 and sample 7, 8, 9 and 10 is compared to sample 6.

Sample	p value	Statistical significance
2	$p = 0.968$	N
3	$p = 0.001$	Y
4	$p = 0.310$	N
5	$p < 0.001$	Y
7	$p = 0.892$	N
8	$p < 0.001$	Y
9	$p < 0.001$	Y
10	$p < 0.001$	Y

Table 5.3: Statistical analysis of navigation results

6 Discussion

6.1 HTTP2 push

For both platforms the application loaded slower ($p < 0.001$) than the non optimized version. The difference between mean times of the non optimized version and the version with HTTP2 push is on desktop 158% and on mobile 144%. This difference is not realistic, instead the most probable explanation is that there is an error somewhere in the implementation of the HTTP2 push functionality. As all load time tests are performed with caching disabled there is no risk of pushing files that are already cached in the browser therefore no additional data is transferred in the HTTP2 push scenario. The expected result would be that HTTP2 push saves one or two RTTs but in the worst case performs the same as the non optimized version.

Due to time restrictions it was not possible to fully understand this result. A lot of variables were checked by verifying the result with different software like the browser, the HTTP2 library, the implementation in the application and other variables were checked by verifying them using debug tools like the pushed files not being transferred multiple times and yet others were checked by comparing the transfer to the non HTTP2 push versions like, the size of the transfer, the transfer encoding, the cache settings. However the result could neither be verified as being or not being affected by an unintentional variable in time.

Due to the uncertainty of the results these can not be considered to answer the research question.

6.2 Code splitting

The version with code splitting did not show a statistically significant difference on the laptop ($p = 0.199$) but it did on mobile ($p = 0.001$). This is probably due to a combination of a slower network that benefits more from a smaller transfer size as well as a slower CPU benefits more from the reduced parse and execution work needed by the smaller code size. Mobile also showed a significant difference between the normal non optimized version and the non optimized version with extra code size ($p < 0.001$) which seems to show that code splitting has a bigger impact the more the application grows, assuming that the growth can be mostly limited to the part(s) that can be split out and loaded under run time. Desktop did not even show a significant difference between the version with code splitting and the non optimized version with extra code size ($p = 0.152$).

When navigating to a different page both platforms showed a significant increase in time ($p = 0.001$ on desktop and $p < 0.001$ on mobile) by having to download the split out bundle.

To answer the research question, for desktop, code splitting has no meaningful effect on the load time. For mobile, code splitting has a positive effect to a degree that varies with the size of the application from medium to large for average to above average code size.

6.3 Isomorphic

The isomorphic version has a significant decrease in time to first meaningful paint ($p < 0.001$) but has no significant difference on the time to a fully loaded page ($p = 0.473$ on desktop and $p = 0.298$ on mobile). On desktop the mean time to first meaningful paint is 2.3 times longer in the non optimized version compared to the isomorphic version and on mobile it is 9.8 times longer. As having pixels on the screen makes the user perceive the load as being faster [4] this is a big improvement for all applications, but for applications where the user reads the displayed information and then may have to take a few seconds before interacting with the application it can basically be considered a big drop in the noticeable load time.

To answer the research question, making an application isomorphic has a positive effect to a large degree if the big reduction of time to first meaningful paint can be leveraged by the application.

7 Conclusion

The collected data seems to show that making an application isomorphic can reduce the time to first meaningful paint, but not the time to a fully loaded page of a single page web application on both desktop and mobile.

Code splitting can reduce the time to first meaningful paint and the time to a fully loaded page, but it seems to only provide a significant benefit on mobile. On the laptop it did not even show an improvement between the version with code splitting and the non optimized version with extra code size which has about twice the amount of JavaScript, leading to the conclusion that code size should be a low priority concern to applications that only serve desktop users. Having to download the additional bundle when needed was shown to have a significant impact on the navigation time, so implementations of this technique may want to either download the bundle in the background or only splitting out code for rarely visited parts of the application.

The data from the HTTP2 push measurements cannot be trusted and thus no conclusions can be drawn from either the HTTP2 push, or the combined data. Despite of that, one may assume that code splitting and isomorphism can be combined to improve both the time to first meaningful paint and the fully loaded page as code splitting improve the JavaScript transfer, parse and execution time while isomorphism removes the need for JavaScript for generating first meaningful paint.

7.1 Future Research

Nothing can be said about the cost of implementing the researched techniques based on this research. To be able to make recommendations whenever or not the value provided by the researched techniques outweighs the cost, the cost itself must be researched over a larger number of web applications.

As the data from the HTTP2 push measurements cannot be trusted, new research on its effect on load time should be conducted. Changes that could be done in future research on HTTP2 is to test on a different platform than Node.js and/or a different operating system than GNU/Linux, or make sure that there is more time available to understand potentially unexpected results.

References

[1] "Single-page application", Wikipedia, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Single-page_application. [Accessed: 21- Jun- 2016].

[2] R. B. Miller, “Response Time in Man-computer Conversational Transactions,” in Proceedings of the 1968 Fall Joint Computer Conference, Part I, 1968.

[3] J. Nielsen, “Response Times: The 3 Important Limits”, Nngroup.com, 1993. [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/. [Accessed: 03-Mar-2016].

[4] F. Nah, “A study on tolerable waiting time: how long are Web users willing to wait?” , 2004.

[5] P. Kortum, C. Tossell, L. Zhong, A. Rahmati and C. Shepard, “Characterizing Web Use on Smartphones”, Me & My Mobile, 2012.

[6] Httparchive.org, "HTTP Archive - Trends", 2016. [Online]. Available: http://httparchive.org/trends.php?s=All&minlabel=Nov+15+2010&maxlabel=Feb+1+2016#bytesJS&reqJS. [Accessed: 14-Feb-2016].

[7] comScore, "The U.S. Mobile App Report", 2014. [Online]. Available: http://www.comscore.com/Insights/Presentations-and-Whitepapers/2014/The-US-Mobile-App-Report. [Accessed: 03-May-2016].

[8] R. A. Netravali, “Understanding and Improving Web Page Load Times on Modern Networks”,2012.

[9] C. Wild, "The Wilcoxon Rank-Sum Test", University of Auckland, 2016. [Online]. Available: https://www.stat.auckland.ac.nz/~wild/ChanceEnc/Ch10.wilcoxon.pdf. [Accessed: 21- Jun- 2016].

A Appendix 1

Raw results from the measurements, all values are in milliseconds (ms)

Sample	First meaningful paint	Fully loaded	Navigation
5	829	2164	422
5	1220	2690	1770
5	874	2307	437
5	1300	2780	1828
5	1197	2710	770
5	875	2308	365
5	925	2357	411
5	786	2298	416
5	1310	2830	2241
5	849	2236	402
5	893	2242	393
5	865	2205	426
5	894	2222	402
5	937	2318	445
5	1270	2700	2094
5	716	2136	410
5	844	2239	420
5	1250	2860	1823
5	929	2250	434
5	813	2153	509
4	552	1280	917
4	443	1080	800
4	531	1090	908
4	476	1070	547
4	411	1080	928
4	544	1442	736
4	692	1717	311
4	560	1600	311
4	670	1624	285
4	670	1475	313
4	610	1567	321
4	684	1750	286
4	701	1706	330
4	644	1677	408
4	663	1642	299

4	557	1571	316
4	664	1650	732
4	691	1807	291
4	627	1673	375
4	638	1516	304
1	1023	1023	786
1	946	946	938
1	1087	1087	898
1	916	916	749
1	932	932	308
1	1495	1495	425
1	1450	1450	305
1	1501	1501	319
1	1395	1395	292
1	1616	1616	301
1	1552	1552	292
1	1753	1753	304
1	1621	1621	358
1	1705	1705	310
1	1516	1516	324
1	1741	1741	306
1	1720	1720	292
1	1693	1693	309
1	1660	1660	297
1	1481	1481	307
3	932	932	873
3	934	934	829
3	922	922	456
3	1009	1009	836
3	929	929	814
3	1294	1294	622
3	1517	1517	427
3	1530	1530	431
3	1519	1519	419
3	1505	1505	451
3	1408	1408	426
3	1500	1500	423
3	1593	1593	417
3	1468	1468	412
3	1536	1536	462

3	1541	1541	416
3	1508	1508	404
3	1630	1630	429
3	1522	1522	415
3	1295	1295	443
2	2241	2241	797
2	2402	2402	568
2	2306	2306	684
2	2441	2441	566
2	2347	2347	672
2	2203	2203	743
2	2207	2207	289
2	2385	2385	302
2	2178	2178	287
2	2244	2244	312
2	2180	2180	312
2	2191	2191	314
2	2458	2458	309
2	2239	2239	290
2	2360	2360	313
2	2379	2379	305
2	2239	2239	305
2	2495	2495	318
2	2103	2103	277
2	1998	1998	318
6	4444	4733	736
6	4341	4609	840
6	4368	4591	651
6	4406	4706	693
6	4425	4694	822
6	4158	4328	707
6	4100	4267	801
6	4135	4297	731
6	4225	4413	777
6	4198	4349	733
6	4123	4219	761
6	4263	4390	764
6	4016	4160	1033
6	4246	4372	946
6	4209	4242	952

6	5175	4278	1067
6	3982	4121	892
6	4128	4277	951
6	4422	4531	1027
6	4158	4262	951
9	906	5071	701
9	732	4895	649
9	682	4607	619
9	642	4808	690
9	619	4638	680
9	599	4328	696
9	664	4420	770
9	640	4084	752
9	702	4206	728
9	589	4005	619
9	591	4155	614
9	657	4405	623
9	578	4144	680
9	653	4115	604
9	553	4026	647
9	414	4416	567
9	673	4381	666
9	556	4040	762
9	626	4192	812
9	560	3978	762
8	3880	4192	1668
8	3958	4269	1486
8	3969	4282	1527
8	4064	4335	1280
8	3936	4209	1521
8	3591	3596	1678
8	3921	4031	1265
8	3597	3723	1341
8	3767	3878	1187
8	3795	3904	1320
8	4233	4366	1230
8	4005	4183	1220
8	3761	3873	1315
8	3836	3938	1294
8	4394	4546	1308

8	4265	4440	1192
8	4185	4322	1624
8	4142	4320	1395
8	4278	4417	1208
8	4334	4490	1382
7	6647	6728	1043
7	6450	6557	810
7	5869	5963	814
7	6383	6579	897
7	6155	6269	768
7	5798	5879	832
7	6563	6537	789
7	5543	5625	819
7	6075	6206	856
7	5139	5220	830
7	6417	6501	811
7	6185	6275	763
7	6485	6499	860
7	6340	6416	779
7	5942	6023	808
7	6379	6480	755
7	6420	6509	792
7	6363	6446	830
7	6102	6191	770
7	6137	6231	648
10	4942	5019	1059
10	5493	5576	1175
10	5275	5352	1038
10	5567	5647	1041
10	5211	5299	1077
10	5334	5441	1087
10	5575	5707	1093
10	5577	5654	1182
10	5077	5152	1068
10	5055	5145	1121
10	5936	6032	1099
10	5783	5879	1034
10	5429	5527	1149
10	4862	4973	1280
10	5045	5165	1201

10	5861	6001	1150
10	5664	5759	1134
10	5913	6006	1178
10	5449	5550	1145
10	5820	5914	1072
12	5376	5451	
12	5376	5477	
12	5215	5300	
12	5307	5387	
12	5420	5500	
12	5270	5383	
12	5259	5341	
12	5270	5361	
12	5181	5265	
12	5502	5604	
12	5527	5624	
12	5413	5510	
12	5534	5663	
12	5522	5622	
12	5533	5647	
12	5420	5537	
12	5487	5649	
12	5421	5528	
12	5473	5571	
12	5601	5700	
11	1552	1574	
11	1483	1508	
11	1454	1473	
11	1458	1479	
11	1421	1436	
11	1497	1412	
11	1441	1463	
11	1496	1520	
11	1468	1483	
11	1448	1478	
11	1495	1511	

11	1445	1462	
11	1479	1496	
11	1454	1479	
11	1558	1587	
11	1576	1600	
11	1508	1535	
11	1627	1654	
11	1614	1630	
11	1579	1598	