

How Can Web Developers Optimise for Lower-Power Smartphones in the Developing World?

Edward George Prince

6151713

Supervised by Dr Norlaily Yaacob

Computer Science, Department of Computing

29th April 2019



Certificate of Ethical Approval

Applicant:

Edward Prince

Project Title:

How Can Web Developers Optimise for Lower Power Smartphones in the
Developing World?

This is to certify that the above named applicant has completed the Coventry
University Ethical Approval process and their project has been confirmed and
approved as Low Risk

Date of approval:

27 February 2019

Project Reference Number:

P87774

Abstract

This paper looks at methods for optimising modern web applications for users with low-powered smartphones. By analysing a testbed application across multiple JavaScript frameworks and bundlers, it is discovered that performance optimisations can be made without sacrificing user or developer experience. This was found using the Google Development Tools to analyse an app built with React, Node.js, Socket.io and Express.js.

Acknowledgements

A special thanks to Dan Prince for his unwavering encouragement and support in this project.

Contents

1	Introduction	6
1.1	Dissertation Structure	6
1.2	Terminology	7
1.2.1	Devtools-Specific Terminology	7
2	Literature Review	9
2.1	Introduction	9
2.2	CPU	9
2.3	Mobile Web Browser	12
2.4	Development Issues	12
2.5	Summary	12
3	Methodology	14
3.1	Introduction	14
3.2	A Brief History and Analysis of JavaScript and ECMAScript	14
3.3	Testbed Application	14
3.3.1	System Requirements	14
3.4	Case Study One: Effect of Bundlers on Performance	15
3.4.1	Bundlers	15
3.5	Case Study Two: Effect of Different JavaScript Frameworks on Performance	17
3.5.1	Frameworks	17
3.5.2	Preact	17
3.5.3	Inferno	18
3.5.4	Hyperapp	18
3.6	Performance Profiling	18
4	Design and Implementation	21
4.1	Hardware Specifications	21
4.2	Testing Matrix	21
4.3	Case 1 - Bundling	21
4.3.1	Prediction	22
4.4	Case 2 - Framework Analysis	22
4.4.1	Prediction	22
4.5	Summary	22
5	Evaluation and Results	23
5.1	Case Study 1: Bundlers	23
5.1.1	Results of Webpack Bundling	23
5.1.2	Results of Rollup Bundling	24
5.2	Case Study 2: Frameworks	24
6	Discussion	26
7	Project Management	27

7.1	Summary and Methodology	27
7.2	Timeline	27
7.3	Communication	27
7.3.1	Presentation Feedback	28
7.4	Legal, Social and Ethical Implications	28
7.4.1	MIT License	28
7.4.2	Impact of Better User Experiences in the Developing World	28
8	Conclusion	29
9	Bibliography	30
10	Appendices	32
10.1	Appendix A: Evidence of Meetings	32
10.2	Appendix B: Evidence of Presentation	38
10.3	Appendix C: Evidence of Presentation Feedback	47
10.4	Appendix D: Experiment Results	49
10.5	Unbundled Profiling	49
10.5.1	At No Throttle	49
10.5.2	At 4x Throttle	49
10.5.3	At 6x Throttle	50
10.6	React Webpack	51
10.6.1	At No Throttle	51
10.6.2	At 4x Throttle	51
10.6.3	At 6x Throttle	52
10.7	Appendix E: Automation Code	53
10.8	Appendix F: Project Diary	54
10.8.1	First Entry	54
10.8.2	Current Progress	54
10.8.3	Script	55
10.8.4	Testbed Factorial	55
10.8.5	Bundler Implementation	55
10.8.6	Rollup	56
10.8.7	Frameworks	56
10.8.8	Data Collection	56

1 Introduction

When developing for mobile phones, the devices to which web apps are optimised are the current standard of modern mobile phones. Whilst this proves useful for a target audience of users able to buy this standard of device - it is less so for the many people who are unable to access web apps on new smartphones. Much of the developing world uses mobile phones several generations behind what is in popular usage across developed countries. With Google, Mozilla, Microsoft and Huawei all releasing cheap smartphones for developing countries (Hiscott 2016), it is likely the developing world's mobile device population will follow a trend of being considerably less powerful than the mobile device population of developed countries (Hiscott 2016). With over 98% mobile phone adoption in developing countries (Sharwood, 2017) this offers a large market of potential users for web applications.

With the anticipation of a high speed network and low-power CPU future (more detail in chapter 2), there is a strong possibility that developers will want to develop for lower-powered CPU's. To simulate these devices, this research will include analysis of performance changes when the CPU power is being throttled.

Older devices also means a lack of compatibility with the newest browsers, meaning that for global usage, developers must make their applications work on older versions of browsers. To mitigate a loss of quality of the application, the tests should be performed on an application built with modern front-end frameworks, as a realistic benchmark. Then metrics can be gathered by measuring performance of applications even when converted into older versions of JavaScript that are compatible with older browsers - and making sure the application runs on those versions.

In summary - the applications must be browser compatible for older browsers, and should be made with modern JavaScript frameworks and techniques, as not to hinder potential developers.

These objectives will be tested by using the Google Chrome Devtools to monitor the CPU-throttled performance, and different technology implementations that allow comprehensive browser compatibility. There is an expectation that the lighter front-end frameworks perform best.

1.1 Dissertation Structure

The Literature Review explores a selection of the relevant existing literature surrounding and encapsulating the topic, establishing its context. Due to the subject being in its infancy in relation to most areas of Computer Science - there is a distinct lack of relevant literature. Network-based optimisations are a popular article subject - but as it bears little need in developed countries - optimisations for low-powered CPU's is not a thriving research topic.

The Methodology chapter lays the ground work for the experiments to be conducted. It shows the techniques used to gain data, explains how they work, and justifies their selection for this project. The experiments are split up into component sections, each identifying a different area for potential optimisation. Each section has a brief summary, a prediction of the expected results upon modification and re-profiling, and the actual results, how the compare, and why they might differ to the predictions.

The Design and Implementation chapter describes exactly how the experiments were conducted, along with the results. The experiments are split into two case studies, analysing bundler performance, and analysing front-end JavaScript framework performance.

The Discussion chapter begins to weave the results into an argument which responds to the initial research question, along with providing assessment on the quality and validity of this research.

The Project Management chapter provides clarity and reflection on the process of creating this project, and the methodologies employed to ensure completion within the timeframe given.

The Conclusion highlights the extent to which the findings provide a suitable answer to the initial research question. It also highlights further areas for study regarding this research.

1.2 Terminology

Term	Definition
0x	No CPU throttling being applied
4x	Four times CPU throttling being applied
6x	Six times CPU throttling being applied
Devtools	Google's suite of developer tools built directly into Chrome
CDN	Content Delivery Network
ES5	ECMAScript 5
Polyfill	Some
Babel	Compiler

1.2.1 Devtools-Specific Terminology

Metric	Description
Load time	Total load time for page resources to be rendered
TTI	Time to interactive (e.g When the user can interact with page)
Page Weight	Accumulated total file size
Loading	Parsing, sending/receiving requests

Metric	Description
Scripting	Compiling and evaluating scripts
Rendering	Executing page layout
Painting	Drawing the render to the screen
Other	
Idle	Time spent waiting

2 Literature Review

2.1 Introduction

As of 2018, there were 3.7 billion mobile devices connecting to the internet (Stevens, 2018) showing global usage and coverage, and there's nothing to suggest that the growth in mobile usage will do anything other than increase.

Given that we know 90% of users stop using an app due to poor performance, and 86% will uninstall the app altogether (AppDynamics and University of London 2014), there is the possibility for unoptimised web apps to lose huge numbers of potential users due to the app performance not meeting our standard usability criteria.

Nejati and Balasubramanian discuss the difficulty in isolating the effects of the mobile browser in determining the root of poor performance (Nejati, Balasubramanian 2016). This project will differ from that research by making use of the Google Chrome Dev Tools mobile simulator - taking many of the factors out of consideration, allowing isolation on only changes to the CPU performance. They also identify the issue of mobile browsers being much slower than desktop, as agreed with in other research on mobile optimisations in the cloud era (Wang et al. 2013), a problem given the dramatic increase in mobile devices globally.

2.2 CPU

[Criticism checklist] It is suggested that network round-trip time is a major problem. Whilst this may have been true at the time, due to the increase in network capacity, and the trends shown so far, it could be assumed that time-based network problems become increasingly less relevant, with the hardware-based issues becoming more prominent. Particularly in developed countries, modern smartphones with the latest hardware are readily available, which is why this project focuses on mobile phones in usage in the developing world.

Over the last 20 years, Moore's Law has begun to plateau (Simonite 2016), demonstrated by Figure 1 and hardware capability increases along with it.

Networks on the other hand are continuing to get faster and offer better coverage. Taking Kenya (a developing country) as an example, Figure 2 shows that network speeds are continuing to increase over time.

Together, Figures 1 and 2 allude to the likely outcome of a fast-network, slow CPU future.

"Of all ITU [International Telecommunication Union] regions, the strongest growth was reported in Africa, where the percentage of people using the Internet increased from just 2.1% in 2005 to 24.4% in 2018" (ITWeb, 2018). Containing a large number of developing countries, by looking at Africa as a whole, and Kenya

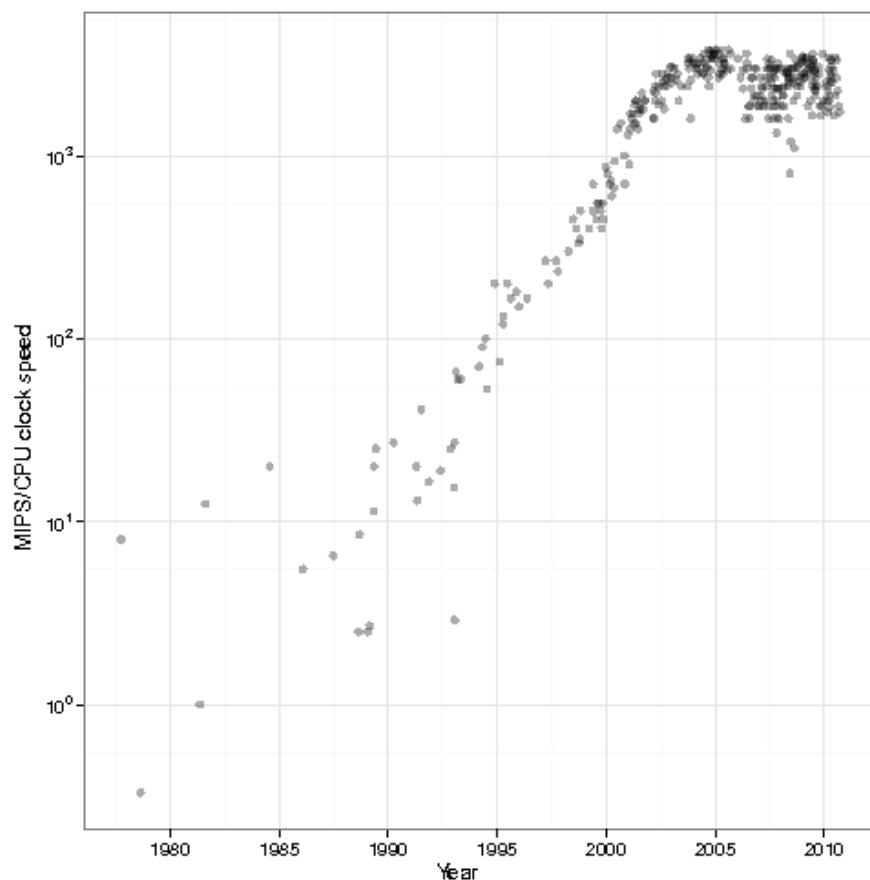


Figure 1: Moore's Law Plateau

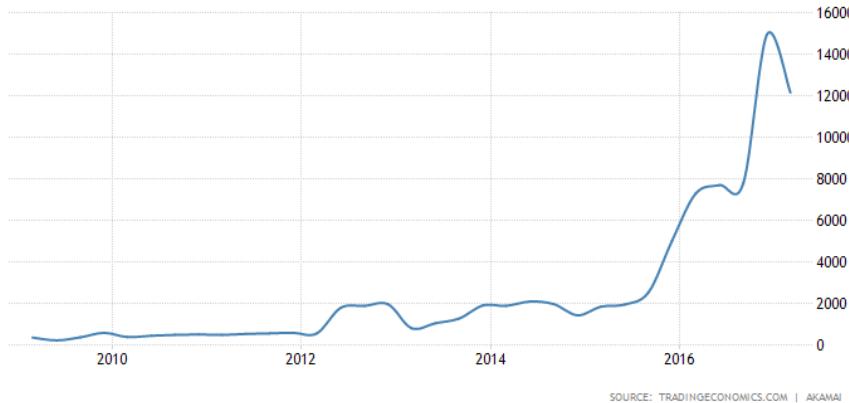


Figure 2: Kenyan Network Speeds

as a microcosm - we can see there is a likelihood of a fast network future - and with the plateau in Moore's Law, a likelihood of stagnation in CPU performance increases.

A fast network future may lead to an emergence in prioritisation of CPU-based optimisation, over decreasing app sizes - which would have an increasingly negligible effect as network speeds improve. There has also been extensive research on optimization over networks already - reducing content, using CDN's, compressing content, utilising browser caching techniques (Iliev 2014), and front end optimisations for modern devices. This research will avoid liminality by relying on the research already conducted on networks, and instead fill the gap left for optimising web applications to perform highly for low-end smartphones. With mobile web browsers being intensive programs for CPU's (Meyerovich and Bodík 2010), there is evidence that optimisations are justified in this field.

Nejati and Balasubramanian provide research around locating mobile browser bottlenecks. They conclude that computational activities are the main bottlenecks, highlighting the need for research on optimising for lower-powered CPU devices, where the effects of computational bottlenecks will be exceedingly more relevant.

It also states that this is not the case for desktops, where network activity is the dominant bottleneck - also emphasizing the importance of optimising CPU-intensive applications for mobile devices. Wang et al. suggest that the most costly area for TTI is resource loading - which, as the sources are credible, makes sense - however no speculation is offered about the state of the future of processors and networks, making the research presently relevant, but perhaps not in years to come.

In summary - the browser is an intensive program, and heavy computational

activities are the main bottlenecks.

2.3 Mobile Web Browser

[criticism checklist] There are also many comparisons made between the mobile browser and desktop browser, over the same network speed - finding that the mobile version can take up to nine times as long to load. As they are on the same network - we know that this is therefore down to the hardware and mobile browser - not server-to-client issues. Suggestions have been made that more effort should be put into making mobile browser optimisations - with which future developers could utilise to make web applications more globally usable.

2.4 Development Issues

Thiagarajan et Al. show that reducing JavaScript and utilising HTML functionality leads to a less power-hungry application. JavaScript being the most power-hungry part of the traditional web application. It also shows the importance of optimising CSS rules, with a reduction to only necessary rules, along with the migration of multiple CSS files into one file for a given page seeing a drop of 5 Joules from the power supply.

It also looks at the benefits of web page prefetching by predicting which links the user will click, and prefetching/loading the relevant data to render the page should the link be clicked more quickly. Does low CPU capability make this a relevant point or not?

The other large aspect they discuss, is the usage of cloud-based architecture. This might increase network latency, and there are other potential security issues, but focusing purely on optimisation, this style of architecture allows processing to be partially taken away from client devices, reducing the workload of the client CPU's. This could be introduced into the chat web app by analysing the impact of moving some intensive functionality from the client to the server.

To keep this research focused into a more specific topic area - architecture optimisations will not be analysed. The topics will be strictly front-end focused.

In summary - whilst prefetching and cloud-based architecture are useful avenues for exploration with regards to mobile optimisation, this research will focus on JavaScript reduction to help lower the load on the CPU.

2.5 Summary

Optimisation over networks is a popular topic, with an array of literature covering the subject. This helps guide this research fill the gap in front-end optimisations. Following on from the research in “Analysing Mobile Web Browser Energy

Consumption”, this project will look at ways to reduce JavaScript by using Bundlers, and a comparison between two different bundlers.

There is a distinct lack of literature based around front-end JavaScript frameworks, which, along with the bundlers creates the foundation of this research. The following chapter will outline how an experiment is created to test the performance of a testbed web application whilst using different technologies.

3 Methodology

3.1 Introduction

In order to perform in-depth analysis on the largest areas for optimisation, a web app would need to be studied in depth to gain insights. Rather than study an existing application, over which no control would be granted, a better solution would be to create a testbed application using modern technologies. In tandem with this, Devtools provides accurate and reliable data on browser applications, with the ability to throttle the CPU to judge performance on simulated lower-hardware. This would allow comparisons to be drawn across the different CPU throttle amounts, but with no other parameter changes.

To understand the decision for the case study subjects, first there must be a basic grasp on the history and current usage of JavaScript and ECMAScript.

3.2 A Brief History and Analysis of JavaScript and ECMAScript

JavaScript is a programming language created in 1995. ECMAScript was designed as a way to standardise JavaScript - and was first released in 1997. Versions have been regularly released, with ES6/ES2015 being the current modern standard. For this dissertation, it will be referred to as ES6 for consistency. The issue with ES6 is that not all versions of browsers support ES6, many still only support up to ES5. This means developers are either stuck developing purely in ES5, limiting them to old features, or they can use the Babel compiler to compile their ES6 code into valid and compatible ES5 - ensuring cross-browser compatibility, whilst having use of ES6 features.

3.3 Testbed Application

In order to comprehensively test the technologies being analysed, a testbed application was required. For the purposes of this project, realtime chatting application was created - an application with genuine use.

3.3.1 System Requirements

The testbed application had to be:

- Made with modern JavaScript frameworks
- Browser compatible with older browsers
- Built with realistic dependencies
- Simulate heavy workload

3.4 Case Study One: Effect of Bundlers on Performance METHODOLOGY

- Functional

Initially, the application was built with React, and later re-written several times using different front-end frameworks. Using bundlers meant it could be written in ES6, but compiled to ES5 to run on older browsers. The app utilises Socket.io to communicate with a Node.js server also running Socket.io. On top of this, the app makes use of several other libraries, such as Bootstrap, Moment, JQuery and Faker, giving a partially realistic build to the application. To simulate heavy workload, a factorial function was created, and called upon initialisation with an input of 1000. This gave the application a significant amount of work to do.

3.4 Case Study One: Effect of Bundlers on Performance

Case study 1 analyses the performance differences between two popular bundlers - Webpack and Rollup. These bundlers are being tested using the testbed application to ensure consistency across all other parameters other than the bundler itself.

To understand why bundlers are used - it is important to understand the history of JavaScript, and its current state. JavaScript is a programming language, created in 1995 - and has been standardised with ECMA (European Computer Manufacturer's Association). Whilst the 9th edition of ECMAScript (ES2018) was finalised in June 2018, only modern browsers can read ES2015 and above, meaning that if developers want their applications to be comprehensively compatible with browsers, the application must be written in ES2015. This raises the issue that the newer versions of ECMAScript have introduced many features of the language that offer superior development - meaning developers want to write their code in ES2015+. This is where a compiler like Babel comes in. Babel can take ES2015+ code, and convert it into ES5 code.

3.4.1 Bundlers

To adhere to good programming conventions, developers tend to build applications out of many modules, but this raises more issues. To import a JavaScript file in HTML, you use a tag like so:

```
<script src="/path/to/file.js"></script>
<script src="/path/to/file2.js"></script>
etc.
```

If the application is made up of 200 different files, the browser has to separately send a request for each file, taking a long time to fetch the resources before the user can interact with the full application. This is where bundlers are used. The bundler can take all these files, and bundle them into one JavaScript file - which the browser then requests:

```
<script src="/path/to/bundle.js"></script>
```

3.4 Case Study One: Effect of Bundlers on Performance METHODOLOGY

Now there is only one file request from the browser, and one file to parse, albeit much larger. Bundlers can do much more than just this however - they can also use plugins like Babel to convert ES2015 to ES5, making the application fully browser compatible, along with minification, setting up development servers and more. This leaves us being able to write modular code, in ES2015+, and still run the optimised app on older browsers. For the purpose of this project, two of the major bundlers were chosen to analyse: Webpack and Rollup.

3.4.1.1 Webpack Webpack is the bundler used by the tool ‘create-react-app’, popular amongst developers for quickly getting React Apps set up. Webpack turns each module into its own isolated function scope.

3.4.1.2 Rollup Rollup puts everything into the same function scope. When measuring the performance of Rollup - it is anticipated that the execution time should decrease, as its handles modules in a different style to Webpack (Lawson, 2016).

```
function foo(a, b) {
    function bar() {
        return a + b;
    }

    return bar();
}

let result = foo(1, 2);
```

When the JavaScript engine reaches this section of code, it creates a new function object in memory for `foo()`. When the engine reaches the line where `foo()` is called, it executes the function, which creates a new function: `bar()`. When the function execution finishes, the `bar()` object in memory is destroyed. This means if `foo()` is called 100 times, the `bar()` function is created and destroyed 100 times. This costs processing, making it generally slower than non-nested functions.

In each case, the bundler will:

- Bundle the JS files
- Bundle the CSS files
- Compile the JS into ES5
- Set up a server and serve the project

The configuration file for each bundler is in the Appendix.

3.5 Case Study Two: Effect of Different JavaScript Frameworks on Performance

Case study 2 analyses the performance difference between four different

3.5.1 Frameworks

Table 3: Framework Comparison

Framework	Size	Number of GitHub Stars
React	106.6KB	127,752
Preact	3KB	22,385
Inferno	48KB	13,619
Hyperapp	1KB	16,660

React has been chosen as the benchmark framework - as the current leader in industry, and will be compared to three other frameworks, designed as faster alternatives.

3.5.1.1 React “React is a declarative, efficient, and flexible JavaScript library for building user interfaces” ([Reactjs.org](https://reactjs.org)) It was designed for component-based web development - making it easier for developers to create reusable components for the applications. It uses JSX, which allows usage of HTML tag syntax within components. At the time of writing, React is in use across 735,040 websites (<https://www.similartech.com/technologies/react-js>). As of January 28th, 2018, React had 1,786,699 downloads, a 152.2% increase from the previous year (<https://jsreport.io/javascript-frameworks-by-the-numbers-winter-2018/>).
<https://books.google.co.uk/books?hl=en&lr=&id=NZCKCgAAQBAJ&oi=fnd&pg=PR6&dq=React+js&ots=Kd&sig=GSstOX8Ag0OYJNGM8hT5nNDSIKU#v=onepage&q=React%20js&f=false>

3.5.2 Preact

Preact is a fast alternative to React - using the same ES6 API (Yomi 2017). The only changes from React are trivial, but most can be accessed by making use of `preact-compat` - an additional package that re-integrates removed features. As of 28th January 2018, Preact had 29,385 downloads, a 254.3% increase from the previous year.

3.5.3 Inferno

Like Preact, Inferno is a lightweight JavaScript library that resembles React. It was created to examine analyse whether a UI library could improve web application experience on mobile devices by addresseeing issues around battery drain, memory consumption and performance (Maida 2016). As of 28th January 2018, Inferno had 29,385 downloads, a 276.5% increase from the previous year.

3.5.4 Hyperapp

Hyperapp is a JavaScript framework for building web applications that uses the Elm-inspired approach to state management. GZipped and compressed, the framework is around 1KB (Bucaran 2018).

3.6 Performance Profiling

To collect accurate metrics about the performance of the chat web app, the Google Chrome Development Tools were used. These tools allow for performance profiling, detailed audits for progressive web apps, and information about load times and page weight. Additionally, they have the capability for CPU throttling, which allows visualising the scale of performance difference between unthrottled and throttled CPU with the same application.

Initially, the unbundled version of the application was analysed, with an unthrottled CPU. This gives the initial set of metrics that can be calculated against. These are:

Then a x4 throttle on the CPU was selected, and the again, giving a new set of results. From here, it was possible to calculate the percentage increase from the unthrottled set to the throttled set.

$$\begin{aligned}
 PI &= \text{Percentage Increase} \\
 T &= \text{Throttled CPU result} \\
 U &= \text{Unthrottled CPU result} \\
 PI &= \frac{T - U}{U} \times 100
 \end{aligned}$$

Across the results, this gives the full range of percentage increases between the data with no CPU throttling and the application of the 4x throttle. Then the steps were repeated but with a 6x throttle applied, and once again calculated the percentage increase. Across multiple passes, the metrics with the largest percentage increase upon throttling the CPU was rendering, idling and other. Despite these being the largest percentage increases, it must also be taken into

account the total time they are taking. A 10% increase of a task taking 1000ms will see a bigger performance hit than a 50% increase of a task taking 5ms. Given the results varied, more passes of the test under the same conditions will be conducted in order to provide the average metric.

To gain the data, Puppeteer was used to run a headless browser, navigate to the URL where the app was being hosted. At this point, the client emulation configuration was decided upon and sent to the headless browser. It could then apply the correct emulation, and begin a timeline trace. Once finished, the trace data would be written to a JSON file, and saved with a name of the current timestamp (to avoid any duplicate file names). The script would run the tracing 20 times each for non-throttled, 4 times throttling, and six times throttling. This reduced outliers, and an average for each metric was collected, before compiling into charts. This technique for data collection has not been used in any of the relevant academic papers found in Chapter 2. The code for this script is found in Appendix E.

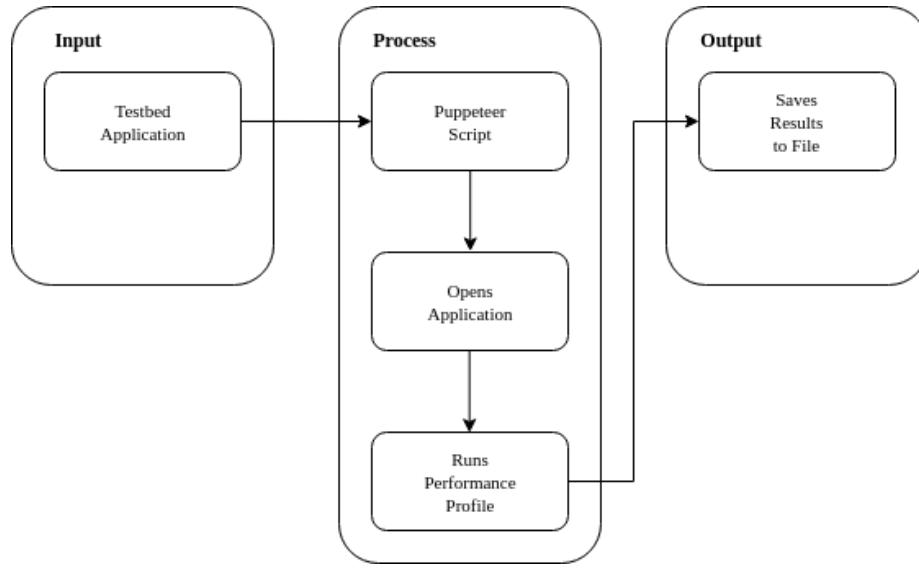


Figure 3: Experiment Flow Chart

As described in Figure 3, each version of the testbed application (e.g React, Hyperapp etc.) would be used as the input - hosting the app locally. Then the Puppeteer script would be run, loading the app and running a performance profile. The output of the script would be written to a file (using JavaScript's `date()` function to create unique file names). Each file could then be loaded into the Devtools and the inbuilt parser would display a summary of key information. This data was then written to a spreadsheet for further analysis.

Webpack supports all ES5-compliant browsers - and if developers wish to support

browsers older than this - polyfills can be loaded to make the application compatible. When using Create React App, webpack's configuration is kept hidden. This default is what will be used in this project to measure performance. This is because the programmer must actively "eject" their application to access customisation of webpack configuration.

Why I chose case 1, 2 etc. (justify with example - expected result) Make a comparison table between predicted/actual results with discussion of relevance
Justify reasoning for cases Requirements flow chart + descriptions

4 Design and Implementation

4.1 Hardware Specifications

The experiment is being carried out on a machine running an AMD FX-8350 with the following specifications:

Table 4: CPU Specifications

Specification	Value
Processor Speed	4GHz
Processor Count	8
Processor Socket	Socket AM3+
Memory Type	DDR3 SDRAM
Wattage	25w

4.2 Testing Matrix

For the final results, each bundler would be tested with each framework about three throttling speeds - 0x, 4x and 6x.

Bundler	React	Preact	Inferno	Hyperapp
Webpack	0x, 4x, 6x	0x, 4x, 6x	0x, 4x, 6x	0x, 4x, 6x
Rollup	0x, 4x, 6x	0x, 4x, 6x	0x, 4x, 6x	0x, 4x, 6x

Using this final matrix, the tests were run with the hardware specified in table 4. Each combination of framework and bundler was tested 20 times separately in the headless chrome browser. This eliminates the possibility of an outlier being used if a single measurement is taken.

4.3 Case 1 - Bundling

To test each bundlers performance, the testbed application was bundled using the configuration files in the appendix (Rollup configuration in Appendix G, Webpack configuration in Appendix H). Once the application has been bundled and served, the Puppeteer scripts found in Appendix E was used to run the Devtools profiling.

4.3.1 Prediction

Upon bundling the app, there are expectations that the loading time reduces, along with the total page weight. There is anticipation that the rendering and painting times will not fluctuate much if at all - as the same content is still being calculated, and exactly the same page is still being painted onto the screen. Loading and scripting times are expected to be where the fluctuation will be seen. It is expected there will be a tradeoff, and the final score will depend on the weighting of that tradeoff. Whilst the browser must do some extra work when parsing the bundled version of the scripts, it also sees the performance benefits as described in section 3.4.1. This also backs up the prediction that of the two bundlers, Rollup should yield better performance.

4.4 Case 2 - Framework Analysis

With the application re-written in four different frameworks - React, Preact, Inferno and Hyperapp, the testing required using each version along with each bundler, and running the Puppeteer script on the resulting locally hosted application.

4.4.1 Prediction

It is expected that React will be the slowest - with all of the other frameworks marketing themselves as faster alternatives to React. With Hyperapp being the smallest of all of them, there is a marginal weighting to expect Hyperapp to be the best performer.

4.5 Summary

Before analysing the results, there are two key predictions - that Rollup should outperform Webpack - and Hyperapp is expected to be the best performing framework. This creates a concise summated prediction statement.

The optimal performance of the testbed application should be seen using the combination of Hyperapp and Rollup.

5 Evaluation and Results

To find an overall score for each combination, the 20 results collected were averaged to create a single score, before the average at each throttle threshold were accumulated to give a final overall score. This represents the overall time the CPU is processing. These final scores can then be compared to the other combinations.

5.1 Case Study 1: Bundlers

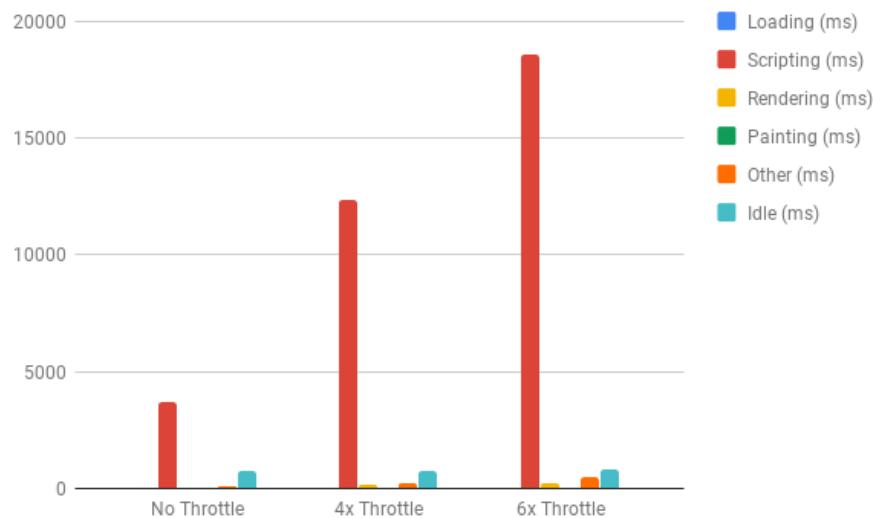


Figure 4: Unbundled Chart

5.1.1 Results of Webpack Bundling

Table 6: Webpack Results (cumulative ms)

	React Webpack	Preact Webpack	Inferno Webpack	Hyperapp Webpack
0x	3553.43	3207.11	3158.77	
4x	12461.31	12307.86	12454.80	
6x	18414.10	18133.34	18261.34	
Total	34428.84	33648.31	33874.905	

5.1.2 Results of Rollup Bundling

Table 7: Rollup Results (cumulative ms)

	React Rollup	Preact Rollup	Inferno Rollup	Hyperapp Rollup
0x	3099.51	3057.97	3081.70	
4x	12088.92	12077.82	11947.46	
6x	17711.29	17624.91	17694.07	
Total	32899.72	32760.7	32723.23	

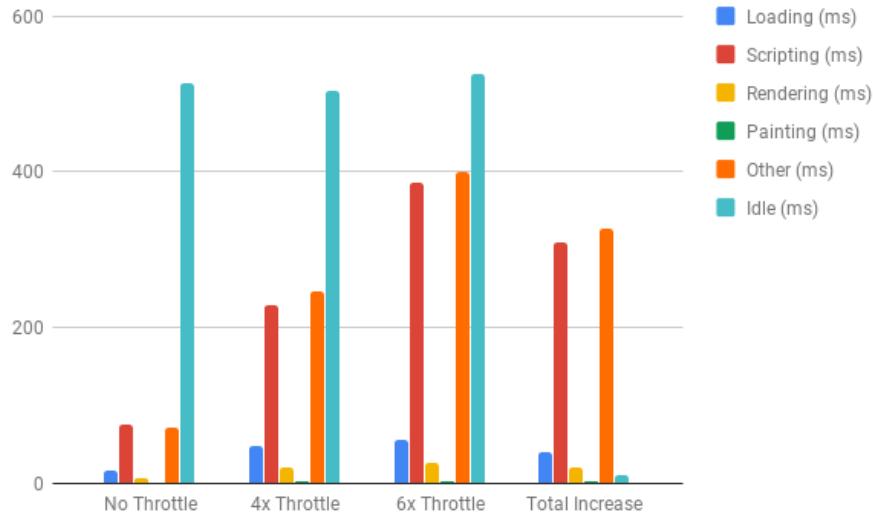


Figure 5: Rollup Chart

It is evident from profiling the application that has been bundled with Rollup has significantly reduced scripting times.

Figure 6 demonstrates that Rollup performs better than Webpack consistently in these tests, with between one and three seconds faster cumulative times over the three throttle thresholds.

5.2 Case Study 2: Frameworks

The framework performance can be analysed using exactly the same data, but for ease of visualisation, the axes have been swapped, and the range has been appropriately shifted and scaled. Figure 7 shows that regardless of the bundler used, Hyperapp has the best performance of all four frameworks.

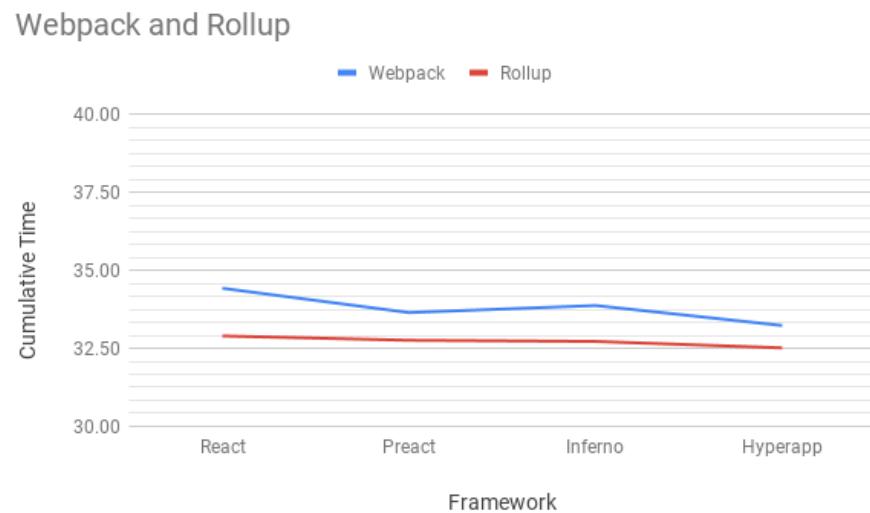


Figure 6: Webpack vs Rollup

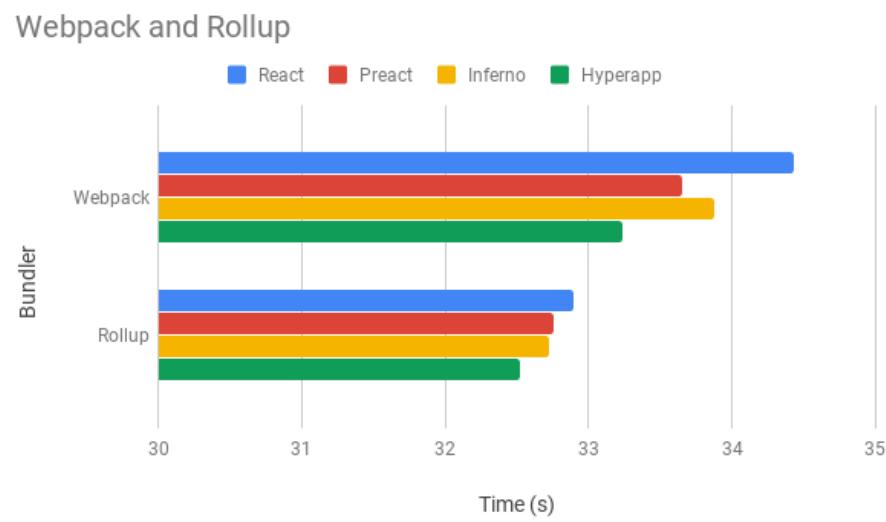


Figure 7: Frameworks

6 Discussion

Talk about service workers as a possible improvement on this research. Testing in other browsers, and testing across more machines.

Whilst useful data has been recorded in this project, the data could be substantially backed up with more experiments. Critically, testing the application across real mobile phones would further the claim that the conclusions were valid in specific relation to the original research question. Building on this - the experiment could also be carried out on specifically the most common devices being used in developing countries, with direct comparisons to performance on new smartphones.

Another useful avenue for testing could be to utilise a more substantial open source application, for example the Hacker News clone on GitHub at <https://github.com/vuejs/vue-hackernews-2.0>. This would also reinforce the conclusions being made in this dissertation, with no uncertainty based on the validity of the testbed application as a source of potential outlying results.

7 Project Management

7.1 Summary and Methodology

A project with multiple complex elements requires careful planning and preparation to be successful, and avoid wasted time in the future. Whilst generally proceedings went smoothly, there were definite lessons learned from this project that can and will be applied to future endeavours. To build the testbed application - the iterative life cycle methodology was employed, allowing the application to quickly progress in the early stages, and gradually improve throughout the entire process. This is due to the acceptance that with an iterative lifecycle methodolgy there will be a lot of rework, particularly at the early stages of the project (Krutchen 2000). The testbed application was also built making use of version control (Git).

With additions like the alternative frameworks, and the automated data collection script, making use of Puppeteer and the Devtools Web API - this method was effective, particularly for a one-person team. (This is where many of the other project management methodologies struggle to have an impact). With the iterative lifecycle methodology being applied at both a micro and macro level, the kanban approach was employed to supplement it, utilising the GitHub Projects interface.

7.2 Timeline

Making use of Gantt Chart, and GitHub Projects feature to plan a schedule. Retrospectively, the project should have been planned in greater detail, allowing the Gantt chart to be less abstracted initially - making definitive goal setting more achievable. Alongside this, the supervisor meetings were utilised to draw up goals and criteria to have achieved by the next session. This allowed the project to be split into weekly chunks that were integrated into the continually evolving Gantt Chart - laying out a clear and concise overall plan for project completion. Supplementing the supervisor sessions, an (almost) weekly diary was kept to track project progress, available in Appendix F.

7.3 Communication

Whilst it took longer than planned to contact the project supervisor - a recognised mistake from the author, when meetings became a regular occurrence, the project began to build steadily and effectively. The supervision was of pivotal importance in relation to this project, with a prediction of difficulty progressing effectively during the early stages.

Evidence of the meetings with the supervisor is located in Appendix A.

7.3.1 Presentation Feedback

After the presentation, the decision was made to change the focus of case study 2. Whilst partially relevant - the research would have shown that web apps perform better with heavy processing done on a server, leaving the client with a light load, and that a server could send a more optimised bundle for a device. Whilst achievable as an experiment - this research would likely not have shed any new light on the area - therefore the JavaScript frameworks comparison became the new subject for case study 2. Alongside this, whilst the topic for case study 2 had changed, the feedback on the justification was still valid - therefore this has been ammended in section 3.6. In addition to this, Figure 3 was created to help the reader visualise the experimentation process.

7.4 Legal, Social and Ethical Implications

7.4.1 MIT License

All of the technologies being tested in this project are open source, and all licensed under the MIT license. This is one of the most permissive licenses, and means that the software created with these licenses can

Whilst now under the MIT license, it is worth noting that React used to be under the BSD+Patents license. which states the code is copyrighted by Facebook (the creators of React). If you sue Facebook for patent infringement, then your patent rights for React are automatically revoked (Kripalani 2017). The switch to MIT proved in response to the disappointment and uncertainty from its community (Wolff 2017).

7.4.2 Impact of Better User Experiences in the Developing World

8 Conclusion

Throughout the experiments put forward in this dissertation, several points can be identified. Firstly, the choice of bundler and framework do impact the performance of a web page - however, depending on the size of the app, the performance benefits may be negligible.

As researched by Lawson, the choice of bundler becomes more integral upon scaling the usage of modules (Lawson 2016). When comparing each of the bundlers over multiple throttling thresholds, it was seen that Rollup was in fact demonstrating its marginal benefits - utilising the shared function scope bundling tactic.

As seen from the full comparison table, the optimal solution when developing for users on lower-power smartphones is the combination of Hyperapp and Rollup. This combination sees the lowest overall score across multiple throttling thresholds.

9 Bibliography

- AppDynamics and University of London (2014) *The App Attention Span Study* [online] available at <<https://www.appdynamics.com/media/uploaded-files/1425406960/app-attention-span-research-report-1.pdf>> [09 February 2019]
- Bucaran, J. (2018) *Hyperapp: The 1 KB Javascript Library For Building Front-End Apps* [online] available from <<https://www.sitepoint.com/hyperapp-1-kb-javascript-library/>> [24 April 2019]
- Hiscott, R. (2016) *The Cheap Smartphones Competing For The Developing World's Internet* [online] available from <<https://www.dailydot.com/debug/cheap-smartphones-for-the-developing-world/>> [22 April 2019]
- Kripalani R. (2017) *If you're a startup, you should not use React (reflecting on the BSD + patents license)* [online] available from <<https://medium.com/@raulk/if-youre-a-startup-you-should-not-use-react-reflecting-on-the-bsd-patents-license-b049d4a67dd2>> [28 April 2019]
- Krutchen, P. (2000) *From Waterfall To Iterative Lifecycle-A Tough Transition For Project Managers* [online] available from <https://www.researchgate.net/publication/237518074_From_Waterfall_to_Iterative_Development_-A_Challenging_Transition_for_Project_Managers> [25 April 2019]
- Lawson, N. (2016) *The Cost Of Small Modules* [online] available from <<https://nolanlawson.com/2016/08/15/the-cost-of-small-modules/>> [17 April 2019]
- Maida K. (2016) *Learn About Inferno JS: Build and Authenticate an App* [online] available from <<https://auth0.com/blog/learn-about-inferno-js-build-and-authenticate-an-app/>> [28 April 2019]
- Meyerovich, L. and Bodík, R. (2010) *Fast And Parallel Webpage Layout* [online] available from <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.207.1244&rep=rep1&type=pdf>> [28 April 2019]
- Nejati, J., Balasubramanian, A., (2016) *An In-depth study of Mobile Browser Performance* [online] available from <<https://www3.cs.stonybrook.edu/~arunab/papers/wprofm.pdf>> [10 February 2019]
- Sharwood, S. (2017) *Developing World Hits 98.7 Per Cent Mobile Phone Adoption* [online] available from <https://www.theregister.co.uk/2017/08/03/itu_facts_and_figures_2017/> [22 April 2019]
- Simonite, T. (2016) *Intel Puts the Brakes on Moore's Law* [online] available from <<https://www.technologyreview.com/s/601102/intel-puts-the-brakes-on-moores-law/>> [03 February 2019]
- Thiagarajan, N., Aggarwal, G., Nicoara, A., Boneh, D., Singh, J., P., (2012) *Who Killed my Battery: Analyzing Mobile Browser Energy Consumption*

- [online] available from <<https://mobilisocial.stanford.edu/papers/boneh-www2012.pdf>> [06 February 2019]
- Wang, H., Kong, J., Guo, Y., Chen, X. (2013) *Mobile Web Browser Optimizations in the Cloud Era: A Survey* [online] available from <<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6525571>> [10 February 2019]
 - Wolf A. (2017) *Relicensing React, Jest, Flow, and Immutable.js* [online] available from <<https://code.fb.com/web/relicensing-react-jest-flow-and-immutable-js/>> [28 April 2019]
 - Yomu (2017) *Introduction to Preact - a smaller, faster React alternative* [online] available from <<https://blog.logrocket.com/introduction-to-preact-a-smaller-faster-react-alternative-ad5532eb6d79>> [28 April 2019]

10 Appendices

10.1 Appendix A: Evidence of Meetings

303COM Record of supervisor meeting

Supervisor: NORLAILY YAACOB

Student: EDWARD PRINCE

Date of meeting: 01/03/2019

Record of individual actions completed + notes:

COMPLETED CHAT APP, FOR DATA COLLECTION

Key topics Discussed:

PROPOSAL FEEDBACK

DISCUSE APP FUNCTIONALITY

CLARIFICATION OF TIMELINE

Individual action points for next meeting (no more than 3)

- To SELECT 3 SUITABLE ARTICLES FOR LIT REVIEW
- PRODUCE A DETAILED GANTT CHART

Date of next meeting: 04/03/2019 10AM

303COM Record of supervisor meeting

Supervisor: NORLAICY Y AACOB

Student: EDWARD PRINCE

Date of meeting: 04.03.19

Record of individual actions completed + notes:

DRAFT OF 3 SUITABLE ARTICLES FOR LIT REVIEW
REVISED GANTT CHART

Key topics Discussed:

FEEDBACK ON LIT REVIEW (FLOW, STRUCTURE)

Individual action points for next meeting (no more than 3)

- FIRST DRAFT OF LIT REVIEW
- LOOK ON TOP500 FOR MORE RELEVANT ARTICLES

Date of next meeting: 11/03/19 10AM

303COM Record of supervisor meeting

Supervisor: NORLAIS YACOUB (Nyacub)

Student: EDWARD PRINCE

Date of meeting: 11.03.19

Record of individual actions completed + notes:

LITERATURE REVIEW CHAPTER

Key topics Discussed:

FEEDBACK ON LITERATURE REVIEW CONTENT
DATA COLLECTION ISSUES - GOOGLE DEV TOOLS

Individual action points for next meeting (no more than 3)

COLLECT DATA FROM GOOGLE DEV TOOLS FOR ANALYSIS
ON PERFORMANCE OPTIMISATION.

Date of next meeting: 18.03.19 12 pm

303COM Record of supervisor meeting

Supervisor: NORLAIRY YAACOB (Maeb).

Student: EDWARD PRINCE

Date of meeting: 18.03.19

Record of individual actions completed + notes:

BROUGHT PERFORMANCE PROFILING RESULTS FROM DEV TOOLS

Key topics Discussed:

METHODOLOGY CHAPTER TO INCLUDE SYSTEM REQUIREMENTS

Individual action points for next meeting (no more than 3)

- PLAN TO PRODUCE AUTOMATED DEV TOOLS PROFILING SCRIPT
- COMPLETE INTRODUCTION CHAPTER

Date of next meeting: 25.03.19 12pm

303COM Record of supervisor meeting

Supervisor: NORCLAY YACOB (Nyacobi)

Student: EDWARD PRINCE

Date of meeting: 25.03.19

Record of individual actions completed + notes:

Extend to case #4.1: Bundling performance analysis

Draft methodology chapter

Key topics Discussed:

- Feedback on case 1
- Document structuring

Individual action points for next meeting (no more than 3)

Case 2: Decide + carry out analysis on selected parameters

Date of next meeting: 01.04.19 12 pm

303COM Record of supervisor meeting

Supervisor: NORLAILY YAACOB (MY/awb) 1/6.

Student: EDWARD PRINCEDate of meeting: 01.04.19**Record of individual actions completed + notes:**

- TESTED APPLICATION
- DECIDE ON DATA COLLECTION STRATEGY
- IMPLEMENT AUTOMATED SOLUTION
- COLLECT DATA ON WEBPACK BUNDLING
- ANALYSE RESULTS AND PRODUCE CHARTS.

Key topics Discussed:

- DISCUSSION ON SUITABLE DECISION TOOLS IN RELATION TO CASE 2:
SERVER-BASED DECISION TOOLS.
- TO INCLUDE TESTING SECTION IN THE REPORT (CASE 1, CASE 2)

Individual action points for next meeting (no more than 3

GATHER METRICS FOR OTHER BUNDLERS AND COMPARE RESULTS

Date of next meeting:

10.2 Appendix B: Evidence of Presentation

The Problem

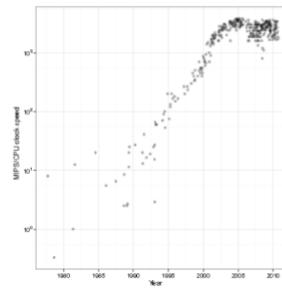
There is research being conducted into many areas of web optimisations, browsers, networks, battery etc. However, there is a distinct lack of research on optimising for low-powered CPU's rather than the current generation of CPU.

Why should there be research conducted?

Whilst predominantly the developed world has less need for web apps that have been optimised for low-powered smartphones - it's not the case globally. With limitations on cost of modern mobile phones - large parts of developing populations use older devices, with a lower hardware specifications than in common usage within the developing world.

Moore's Law

Over the last 10 years, Moore's Law has begun to plateau - and suggests at least in the near future, the little increase in computational power may continue to preside.

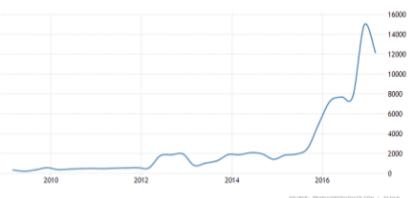


Network Speeds

In contrast to this - there has been a general shift in network speeds increase over the last 10 years in Kenya.

Should these trends continue - this would leave a fast-network/low-powered CPU future.

Kenyan Network Speed



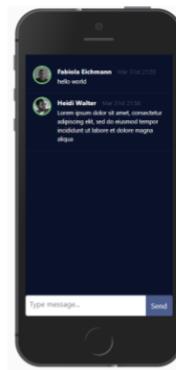
Fast Network/Low Power CPU Future

Carrying Out the Research

1. Create a testbed application
2. Collect data on application
3. Implement a change
4. Collect contrasting data
5. Analyse results

Testbed Application

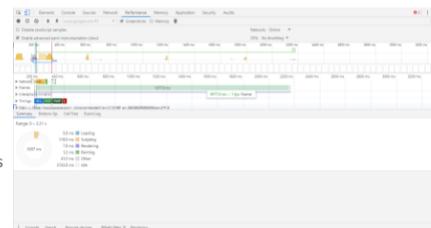
A testbed realtime chatting application - utilising modern industry-standard technologies (React, Node.js, Express.js, Socket.io) was created with which to experiment - granting the ability to gather in-depth metrics upon applying new technologies (webpack, rollup, parcel, server-decision tools).



Google Chrome Devtools

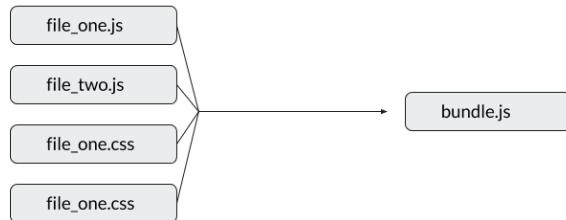
The Google Chrome Devtools is the industry standard set of tools that developers use in the browser to gain insights into how code is interacting with the browser.

There are also tools like Puppeteer to allow headless browser automation, giving easier access to more results within a given timeframe.



Case 1: Bundling an Application

What is bundling?



Data Collection with Chrome Devtools

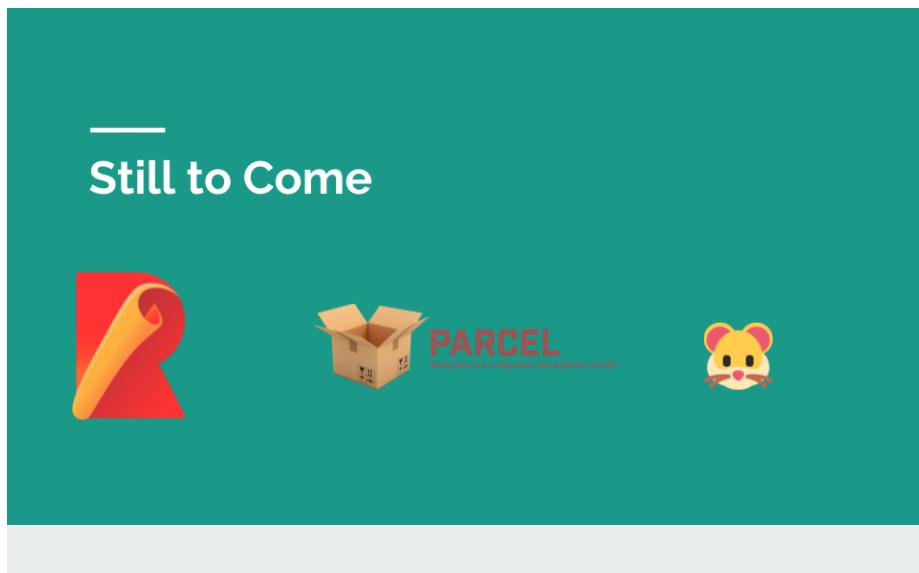


Running the app in its current state through the Devtools performance profiler yields useful data about the load times and performance metrics.

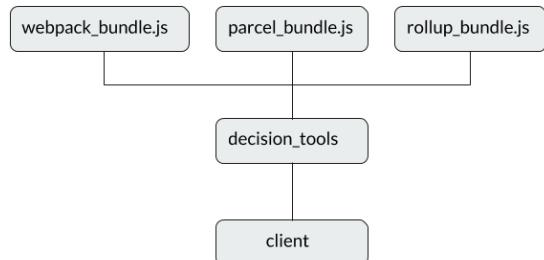
This data can then be scraped and put into a spreadsheet for further analysis.

Using a custom script - much of this process could be automated for faster collection - meaning more results could be collected.





Case 2: Server-Based Decision Tools



Client vs Server-Based Architecture

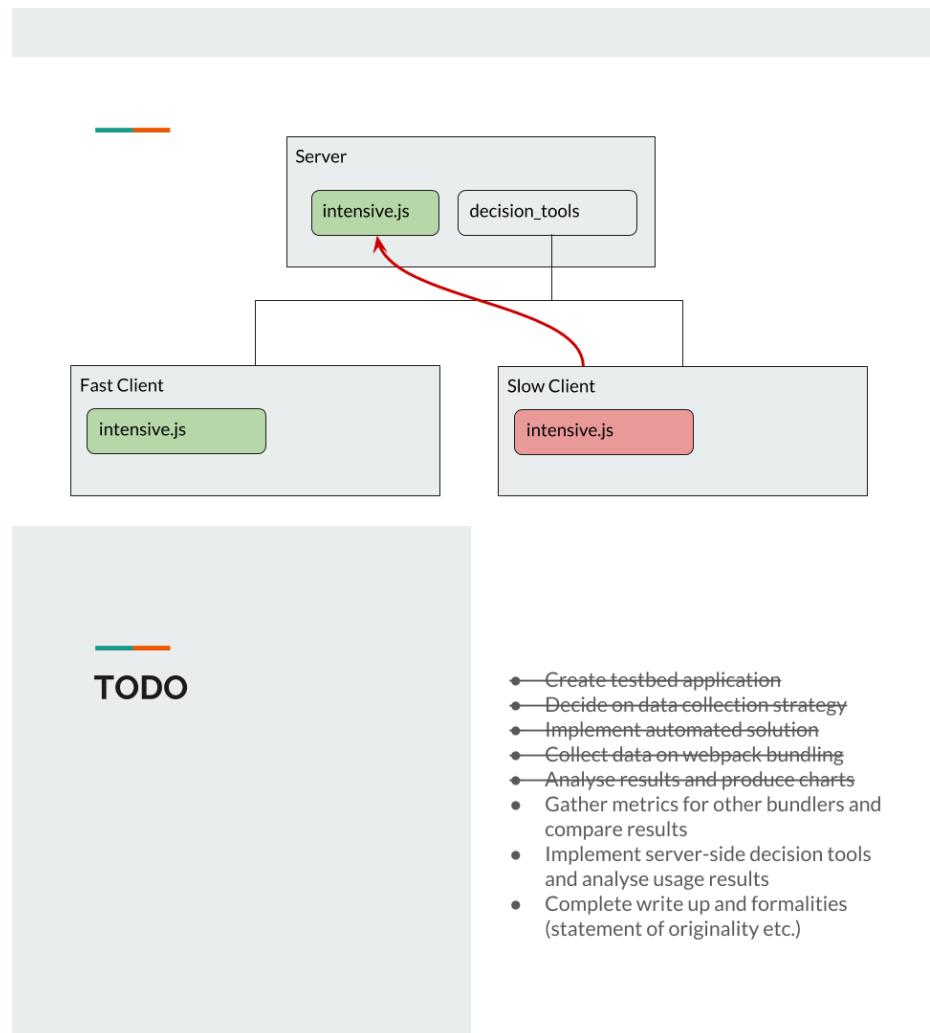
Web technology has seen shifts in desires for architecture to be client and server based. With powerful modern devices - it is becoming more common for the heavy lifting to be done on the client - leaving the server running lightly to keep high speeds with high volumes of traffic.

With low-powered devices, if developers wish for fast user experiences, then the server-based architecture become relevant again.

Accessing the Client

JavaScript is now capable of acquiring an estimate of the processor speed that is executing the script. This means an application can tell the server basic information about the device - allowing the server to make useful decisions on what it should and shouldn't send/process.

This optimises processes for specific devices, keeping servers quick and using client-based architecture where devices are capable - and switching to server-based architecture and optimised bundles for older browsers and slower architecture.



Bibliography

- Gough, N. (2005) *Africa: The Impact of Mobile Phones* [online] available at <<https://www.share4dev.info/telecentreskb/documents/4552.pdf>> [26 January 2019]
- Iliev, I. (2014) *Front end optimization methods and their effect* [online] available from <<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6859613>> [03 February 2019]
- ITWeb (2018) *Africa Sees Strong Internet Growth* [online] available from <<https://www.itweb.co.za/content/VgZevJA3V6qdIX9>> [11 February 2019]
- Simonite, T. (2016) *Intel Puts the Brakes on Moore's Law* [online] available from <<https://www.technologyreview.com/s/601102/intel-puts-the-brakes-on-moores-law/>> [03 February 2019]
- Trading Economics (2017) *Kenya Internet Speed* [online] available from <<https://tradingeconomics.com/kenya/internet-speed>> [10 February 2019]
- Wang, H., Kong, J., Guo, Y., Chen, X. (2013) *Mobile Web Browser Optimizations in the Cloud Era: A Survey* [online] available from <<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6525571>> [10 February 2019]

10.3 Appendix C: Evidence of Presentation Feedback

303COM Record of supervisor meeting

Supervisor: NORLAILY YACOB (Nyacobs) M4

Student: EDWARD PRINCE

Date of meeting: 01.04.19 (PROJECT PRESENTATION)

Record of individual actions completed + notes:

PRESENTATION

Key topics Discussed:

PRESENTATION FEEDBACK

1. PRODUCE AN OVERALL BLOCK DIAGRAM OF DESIGN PROCESS FOR PROPOSED WEB OPTIMISATION TECHNIQUES.
2. TO JUSTIFY THE FUNCTION OF DECISION TOOLS IN CASE 2: SERVER-BASED DECISION TOOLS.
3. TO INCLUDE SOCIAL, LEGAL AND ETHICAL IMPLICATIONS IN RELATION TO THE PROJECT.

Individual action points for next meeting (no more than 3)

Date of next meeting: —

Figure 8: Presentation Feedback

10.4 Appendix D: Experiment Results

10.5 Unbundled Profiling

10.5.1 At No Throttle

Table 8: Unbundled at No Throttle

Run (0x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	4.7	3111.8	43.4	0.6	60.9	453.4
2	4.1	3431.6	48.2	0.7	62.4	5100
3	4.8	3180.1	47.4	0.9	65.6	440.6
4	1	3197.6	50.7	0.4	36.1	156.2
5	4.7	3899.6	77.9	0.7	72.8	539.3
6	7	3834.2	54.9	1.3	88.1	477.8
7	4.9	3941.5	62.4	12.8	83.9	617.5
8	5.3	3694.9	53.2	1.4	94.1	515.8
9	5.2	4164.4	59.1	0.7	84.2	670.3
10	5.7	3434.5	59.3	3	157	804.3
11	4.6	3581.2	55.8	2.6	77.2	682.4
12	4.2	3206.8	45.5	0.6	71.9	524.1
13	5.8	4151.7	53.5	0.7	88.5	500.6
14	4.1	4656.3	59.2	0.8	78.9	555.5
15	12	4185.6	57.1	1.2	80.9	664.8
16	4.2	3698.1	51.7	1.1	84.2	492.3
17	4.3	3724.6	49.5	2.6	78.6	496.5
18	5.4	3618.6	53.3	0.9	83.2	496.6
19	5.4	4018.2	57.1	1	85.8	525
20	4.6	3551.9	50.9	0.7	82.7	490.1
Average	5.1	3714.16	54.505	1.735	80.85	760.155

10.5.2 At 4x Throttle

Table 9: Unbundled at 4x Throttle

Run (4x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	15.5	12827.6	164.2	2.1	186.8	645.7
2	6.2	10566.3	159.9	1.9	214.1	734.5
3	12.9	13499	155.4	2.1	205.1	671
4	19.8	12701.8	155.1	2.2	193.6	898

Run (4x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
5	10.7	9852.5	187.1	6.7	372	713.9
6	11	12833.3	178.4	7.9	261.2	947.5
7	7.2	11398	180.7	6.7	290.4	661.1
8	16.1	12382.4	165.8	2.4	192.2	799.7
9	12.1	12313.2	181.9	2.3	313.2	932.5
10	15.8	14077.1	180.6	2.1	274.5	750.2
11	18.3	7225.1	169.9	2.3	190.4	774.6
12	9.3	13893.1	187.2	2.4	121.5	594.6
13	13.2	13131.3	199.9	5.2	234.9	789.8
14	9.8	14363.4	167.2	1.7	199.9	967.2
15	19	11134.7	168.8	6.6	219	553
16	19.5	12194.9	177.9	3.7	235	627.1
17	11.8	12514.6	187	2.8	186.3	942.6
18	14.6	12724.4	160.5	3	282.4	797.2
19	26.1	15305.5	172.1	3.2	398	816.5
20	16.3	12208.5	176.5	6.3	242.7	584.7
Average	14.26	12357.335	173.805	3.68	240.66	760.07

10.5.3 At 6x Throttle

Table 10: Unbundled at 6x Throttle

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	8.5	19643.1	211.8	2.4	246.7	577.7
2	6.4	14932.7	191.6	2.6	644.9	631.9
3	23.7	19366.7	254.1	3.3	244.1	552.2
4	18.3	19754.3	184.8	4.3	351.3	538
5	11.4	19008.9	227.9	2.9	715.7	668.5
6	25.9	21328.3	108.5	14.9	1622.9	1117.2
7	9.6	14051.8	239.6	3.1	365	1273.9
8	9.2	19393.9	188.9	4.9	334.8	715.2
9	17.5	21682.2	271.2	4.1	529.6	980.9
10	21.3	22173.7	243.6	4	835.8	851.5
11	9.2	17532.4	256	4.4	320.8	921.1
12	8.2	19373.5	251.3	2.7	213.5	654.6
13	8	17301	224.3	1.8	387.6	1423.8
14	11.2	20298.6	118	2.9	512.2	1036.1
15	11.5	16325	232.9	10.1	220.3	797.8
16	9.8	18459.1	178.3	1.4	303.8	684.5
17	22.3	18405.5	380.6	3.9	249.3	852.9

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
18	28.6	19070.7	224.5	1.9	1249.2	562.1
19	15.2	13584.4	177.1	1.1	307.5	794.6
20	10.1	20168.3	260	3.1	319	662.9
Average	14.295	18592.705	221.25	3.99	498.7	814.87

10.6 React Webpack

10.6.1 At No Throttle

Table 11: Bundled at No Throttle

Run (0x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	12.2	2852.9	54.7	1.3	45.9	524.6
2	13.1	3011.8	67.6	1.0	41.6	466.0
3	13.4	2916.5	52.7	1.0	43.3	465.9
4	12.1	2790.0	51.6	0.7	41.5	479.0
5	13.5	2952.6	54.9	0.8	43.3	500.5
6	13.6	2769.9	55.6	0.8	41.6	558.1
7	13.8	2935.2	53.6	0.9	57.3	454.2
8	11.2	2869.7	56.3	0.8	50.4	520.7
9	13.2	3969.4	51.3	0.6	48.9	568.5
10	13.5	2838.4	52.9	0.8	45.2	436.2
11	13	2867	54.7	0.8	42.7	454.7
12	12.8	2826.9	50	0.6	46	492.1
13	12.1	2902.6	50.5	0.9	45.8	490.6
14	12.5	2929.5	50.7	1	46.1	469.7
15	12.2	2960.9	58.4	0.7	43.5	573.8
16	13.2	2844.5	50.3	1.2	55.6	476.6
17	11.9	2934.4	51.4	0.7	42.5	475.6
18	12.5	2965.1	51.5	0.7	43.8	450.5
19	13.2	2968.4	52.2	1.4	48.4	438.2
20	12.7	2930.3	52.6	1	45.1	471.6
Average	12.785	2951.8	53.675	0.885	45.925	488.355

10.6.2 At 4x Throttle

Table 12: Bundled at 4x Throttle

Run (4x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	21	12102.7	180.8	3.4	91.2	638.8
2	55.2	10741.4	186	0.2	228.3	401.6
3	41.4	10910.9	176	1.8	106.8	431.7
4	58.4	12146.9	184.8	0.6	86.3	509.5
5	40.3	10720.5	181.9	0.8	97.9	454.6
6	42.2	11731.6	184.5	1.6	102.2	503.4
7	40.4	11037.4	196	0.6	99.4	495
8	32.3	12096.9	188.9	1.3	78	515.9
9	34.2	11619.8	179.9	0.5	119.5	649
10	38.7	11833.5	174.1	0.5	156.8	529
11	35.2	11706.1	178.4	0.7	131.6	445.2
12	44.7	12340.6	193.7	6.4	132.9	446.4
13	101.3	12052.7	172.7	1	85	422.4
14	64.2	11944.8	190.9	1.3	91.5	542.7
15	51.8	11049.8	179.3	0.4	86.6	482.5
16	45.5	12259.8	187.5	0.3	80.3	430.5
17	47.1	11224.3	182.4	2	172.3	434
18	43.5	11544.7	188.6	1.4	108	405.9
19	26.9	11722.5	184.7	2.3	112.6	529.3
20	34.7	11774.3	181	0.8	120.8	510.6
Average	44.95	11628.06	183.605	1.395	114.4	488.9

10.6.3 At 6x Throttle

Table 13: Bundled at 6x Throttle

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
1	58.7	18068.7	166.6	2.6	198	472.7
2	54.8	18064.4	269.8	0.4	442.1	467.8
3	25.8	16431.1	218.1	7	113.9	482.1
4	51.3	18038	228.6	1	242.2	447.1
5	39.2	16284.8	226.5	5.5	114.6	570.7
6	43.7	18446.5	360.7	3	340.1	355.3
7	64.2	16201.7	234.2	2.2	370.6	559.4
8	32.5	18417.2	255.2	7	179	389.6
9	93.8	16435.9	229.9	1.8	208.8	530.6
10	24	17649.1	216.2	0.4	175.8	470.5
11	88.5	17280.8	216.9	3.3	200.7	422.4

Run (6x)	Loading (ms)	Scripting (ms)	Rendering (ms)	Painting (ms)	Other (ms)	Idle (ms)
12	30.3	17540.1	220.3	1.6	216.1	525.5
13	109.3	18272.8	381.6	7.3	253.3	406
14	46	17913.8	245.8	2.1	250.7	510.1
15	81.5	16256.5	236.4	0.7	122.8	570.3
16	26.4	18408.6	136	0.3	261.7	493.8
17	46.1	16755.5	223.5	0.4	133.4	451.9
18	39.8	17334.5	246	0.9	160.4	454
19	37.8	16258.8	211.5	0.4	95.8	456.7
20	35.8	18704.1	223.3	0.8	199.4	386.3
Average	51.475	17438.145	237.355	2.435	213.97	471.14

10.7 Appendix E: Automation Code

```

const puppeteer = require('puppeteer');
let i = 0;
const DIR = 'hyperapp-rollup';
const URL = 'http://localhost:5000';

let loop = setInterval(async () => {
    if (i === 19) {
        clearInterval(loop);
    }
    throttleSix(i);
    throttleFour(i);
    throttleNone(i);
    i++;
}, 20000)

async function throttleSix(i) {
    const browser = await puppeteer.launch();
    const page = await browser.newPage();
    const client = await page.target().createCDPSession();
    await client.send('Emulation.setCPUThrottlingRate', { rate: 6 });
    await page.tracing.start({
        path: `traces/${DIR}/6x/${i}profile-${Date.now()}.json`
    });
    await page.goto(URL);
    await page.tracing.stop();
    await browser.close();
}

```

```

async function throttleFour(i) {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  const client = await page.target().createCDPSession();
  await client.send('Emulation.setCPUThrottlingRate', { rate: 4 });
  await page.tracing.start({
    path: `traces/${DIR}/4x/${i}profile-${Date.now()}.json`
  });
  await page.goto(URL);
  await page.tracing.stop();
  await browser.close();
}

async function throttleNone(i) {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  const client = await page.target().createCDPSession();
  await page.tracing.start({
    path: `traces/${DIR}/0x/${i}profile-${Date.now()}.json`
  });
  await page.goto(URL);
  await page.tracing.stop();
  await browser.close();
}

```

10.8 Appendix F: Project Diary

10.8.1 First Entry

28/01/2019

I had a meeting with my supervisor today. We discussed current progress and direction. I will be working on producing a more highly detailed Gantt chart to demonstrate how I will work to the project timeline. I am also going to be working on the literature review.

10.8.2 Current Progress

05/02/2019

The testbed web application has been built, utilising socket.io for realtime chatting functionality. I used React (a widely used modern front end JavaScript framework) to build the front end, and Express.js (popular backend Node.js library) for backend programming. I am using GitHub to version control the application, and the server is designed distinctly as its own entity - not tied to the front end at all - meaning they are not reliant on the other to work. To help

isolate the problems - I am going to focus attention on optimising the front end - not the server.

The report is coming along slowly - I now have a bare skeleton of the entire report, with elements from the project proposal being dropped in. I am going to work on some padding out of the literature review I conducted, and try to get a structure for the main literature review sorted.

10.8.3 Script

12/02/2019

I have created a script that uses Puppeteer to open a headless browser with the application running, then automatically perform a trace, with simulated throttled CPU's, and writes the resulting data into a JSON file and saves the file into the repository. I can then load up each of the JSON files into the Chrome Dev Tools, and it produces summary data on the trace. I can then write the results into a Google Sheet to generate averages and compare results. I have also mostly finalised the literature review, and have begun work on the Methodology Chapter.

10.8.4 Testbed Factorial

20/02/2019

In order to make the testbed application somewhat more realistic, I needed to get the app to do some heavy lifting upon loading. To simulate this, I wrote a Factorial calculator function, and made the application solve factorial(1000) upon initialising. This gave a more realistic set of metrics on how the bundlers and frameworks perform at a higher stress load. After the meeting with my supervisor, we finalised what would go into the Methodology, and that splitting into a Methodology Chapter and a Design & Implementation Chapter would allow for a better report. The methodology will now just contain the relevant information about how and why the experiment was conducted in the manner it was - but not detailed information into the experiment. Alongside this, I tried to improve the automated script to not only collect the data, but automatically parse the profiles to output the relevant data. After spending several days researching and attempting an implementation - I decided to move on. This meant that for each performance profile that was collected, I needed to open the file in the Devtools and copy the relevant statistics over into a Google Spreadsheet.

10.8.5 Bundler Implementation

01/03/2019

I have changed from using create-react-app to using a custom webpack configu-

ration file to allow for more comparable results to Rollup. The create-react-app webpack configuration is all hidden, and has a lot of extra stuff that Rollup doesn't do by default. Alongside this, I have been working on the Methodology chapter.

10.8.6 Rollup

08/03/2019

I have now got both Webpack and Rollup implemented so the testbed application can be bundled using either, allowing for comparisons to be made.

10.8.7 Frameworks

17/03/2019

There is now implementations of the testbed application written in React, Preact, Inferno and Hyperapp to allow performance profiling across all four frameworks. Using my automated script, I can make the app compile from any of the frameworks using either of the bundlers, before automatically collecting data on the performance, and saving to a Google Sheet. In addition to this, I have been researching legal and ethical implications of the project - but finding it difficult to find much relevant information.

10.8.8 Data Collection

01/04/2019

I have now collected all of the relevant data for each combination of framework and bundler, and gather metrics for no throttling, 4x throttling and 8x throttling. All the data is saved in Google Sheets to allow for chart generation to help visualize the results. I have also been finalising the Methodology and Design & Implementation chapters - now having all of the information required to finish them.